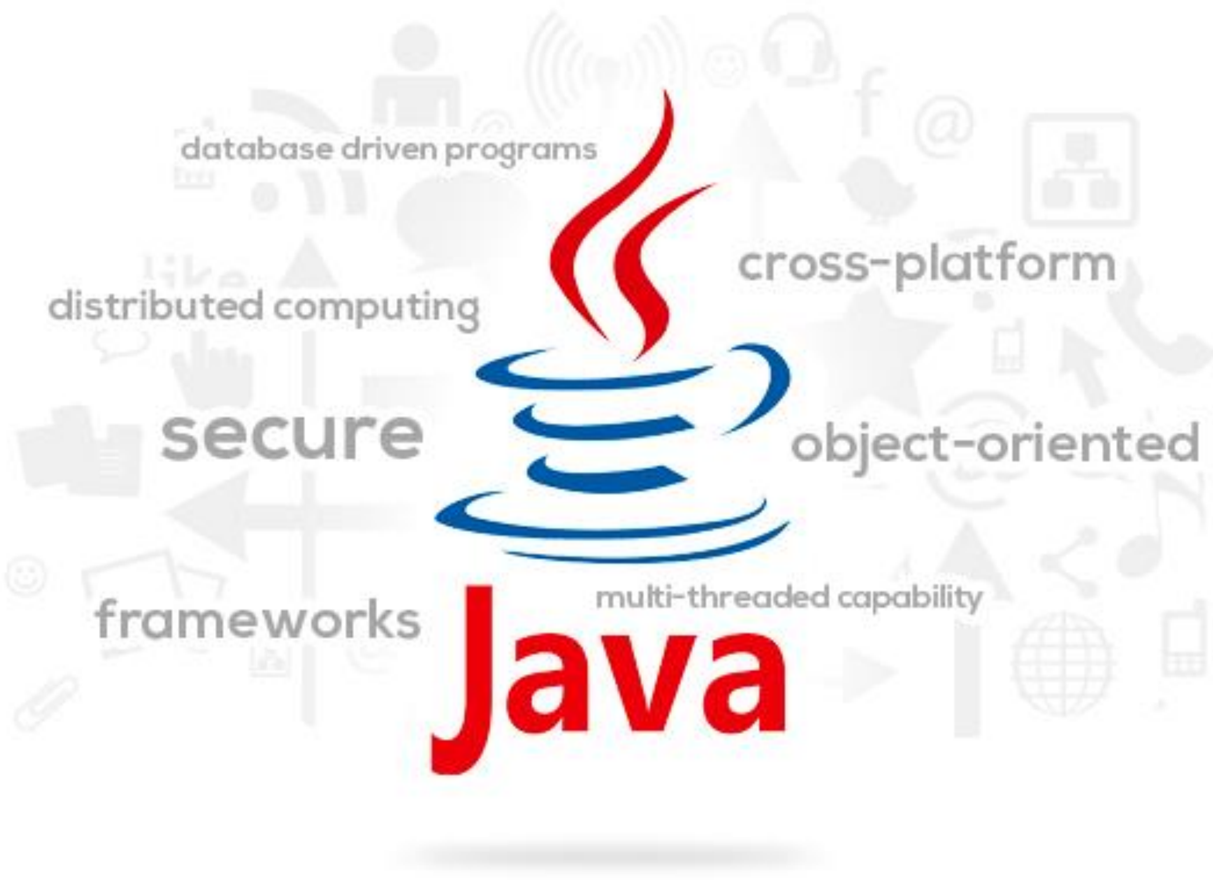


Yadab Raj Ojha

Sr. Java Developer / Lecture
Email: yadabrajojha@gmail.com
Blog: http://yro-tech.blogspot.com/
Java Tutorial: https://github.com/yrojha4ever/JavaStud
LinkedIn: https://www.linkedin.com/in/yrojha
Twitter: https://twitter.com/yrojha4ever
Website: https://www.javaenvagilist.com

Part 4

Collections and Generics

Collections and Generics

Collections

- Hierarchy and methods
- Iterator interface
- List: ArrayList, LinkedList, Vector
- Set: HashSet, LinkedHashSet, TreeSet
- Map: HashMap, TreeMap, LinkedHashMap
- Iterator/ ListIterator

Collections in Java

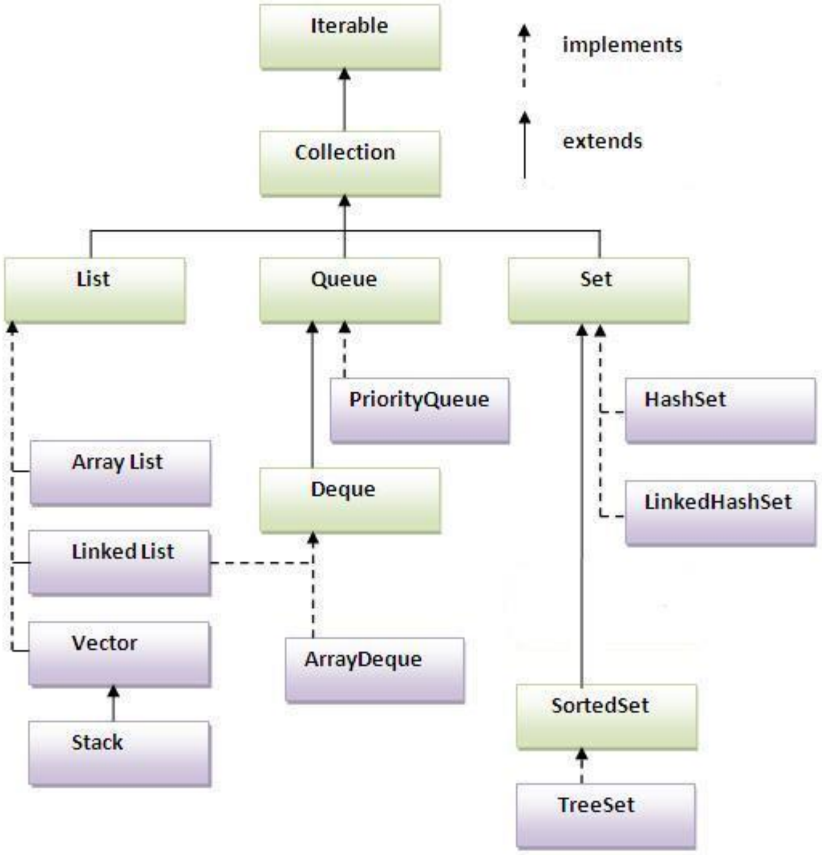
Collections in java is a framework that provides an architecture to store and manipulate the group of objects. Collections are like containers that groups multiple items in a single unit. For example; a jar of chocolates, list of names etc.

All the operations that you perform on a data such as searching, sorting, insertion, manipulation, deletion etc. can be performed by Java Collections.

Java Collection simply means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque etc.) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet etc).

Hierarchy of Collection Framework

Let us see the hierarchy of collection framework.The **java.util** package contains all the classes and interfaces for Collection framework.



Methods of Collection interface

There are many methods declared in the Collection interface. They are as follows:

No.	Method	Description
1	public boolean add(Object element)	is used to insert an element in this collection.
2	public boolean addAll(collection c)	is used to insert the specified collection elements in the invoking collection.
3	public boolean remove(Object element)	is used to delete an element from this collection.
4	public boolean removeAll(Collection c)	is used to delete all the elements of specified collection from the invoking collection.
5	public boolean retainAll(Collection c)	is used to delete all the elements of invoking collection except the specified collection.
6	public int size()	return the total number of elements in the collection.
7	public void clear()	removes the total no of element from the collection.
8	public boolean contains(object element)	is used to search an element.
9	public boolean containsAll(Collection c)	is used to search the specified collection in this collection.
10	public Iterator iterator()	returns an iterator.
11	public Object[] toArray()	converts collection into array.
12	public boolean isEmpty()	checks if collection is empty.
13	public boolean equals(Object element)	matches two collection.
14	public int hashCode()	returns the hashcode number for collection.

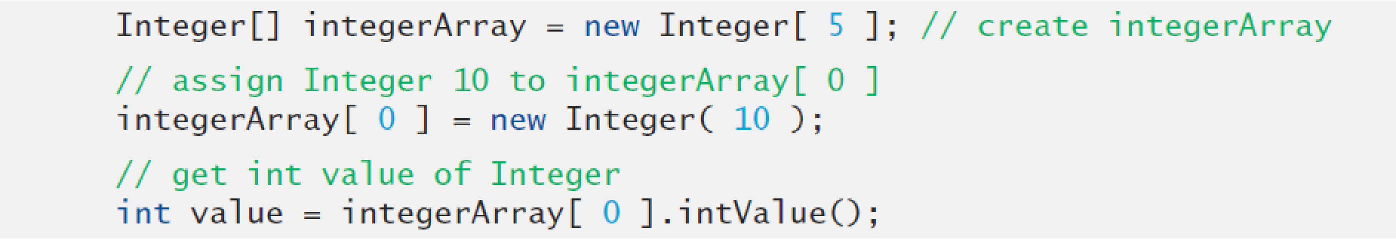
Iterator interface

Iterator interface provides the facility of iterating the elements in forward direction only.

There are only three methods in the Iterator interface. They are:

1. **public boolean hasNext()** it returns true if iterator has more elements.
2. **public object next()** it returns the element and moves the cursor pointer to the next element.
3. **public void remove()** it removes the last elements returned by the iterator.

Prior to Java SE 5, if you wanted to insert a primitive value into a data structure, you had to create a new object of the corresponding type-wrapper class, then insert it in the collection. Similarly, if you wanted to retrieve an object of a type-wrapper class from a collection and manipulate its primitive value, you had to invoke a method on the object to obtain its corresponding primitive-type value.

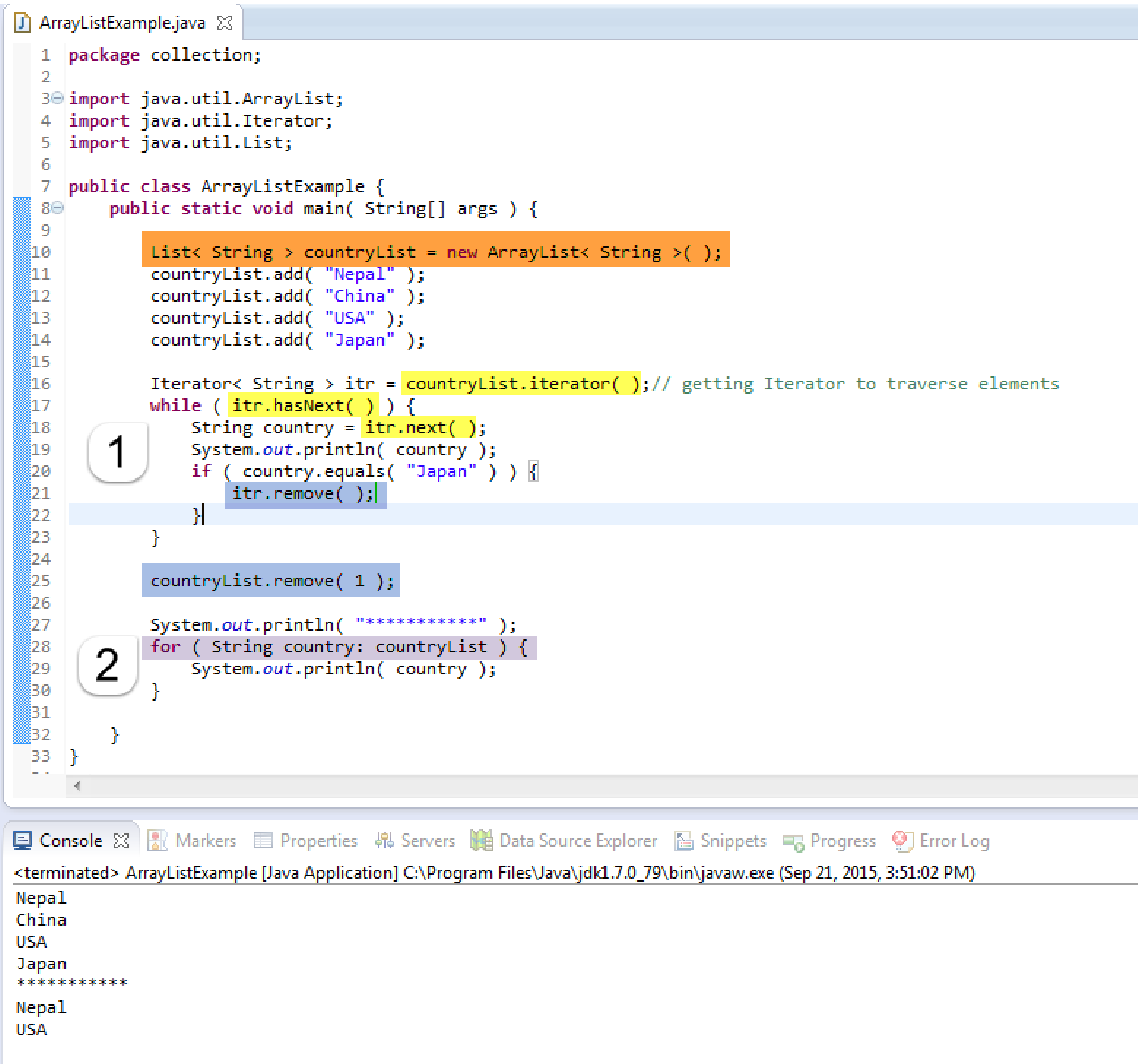


```
Integer[] integerArray = new Integer[ 5 ]; // create integerArray

// assign Integer 10 to integerArray[ 0 ]
integerArray[ 0 ] = new Integer( 10 );

// get int value of Integer
int value = integerArray[ 0 ].intValue();
```

List: (ArrayList, LinkedList, Vector)

Java ArrayList class

- Java ArrayList class uses a dynamic array for storing the elements. It extends AbstractList class and implements List interface.
- Java ArrayList class can contain duplicate elements.
- Java ArrayList class maintains insertion order.
- Java ArrayList class is non synchronized.
- Java ArrayList allows random access because array works at the index basis.
- In Java ArrayList class, manipulation is slow because a lot of shifting needs to be occurred if any element is removed from the array list.

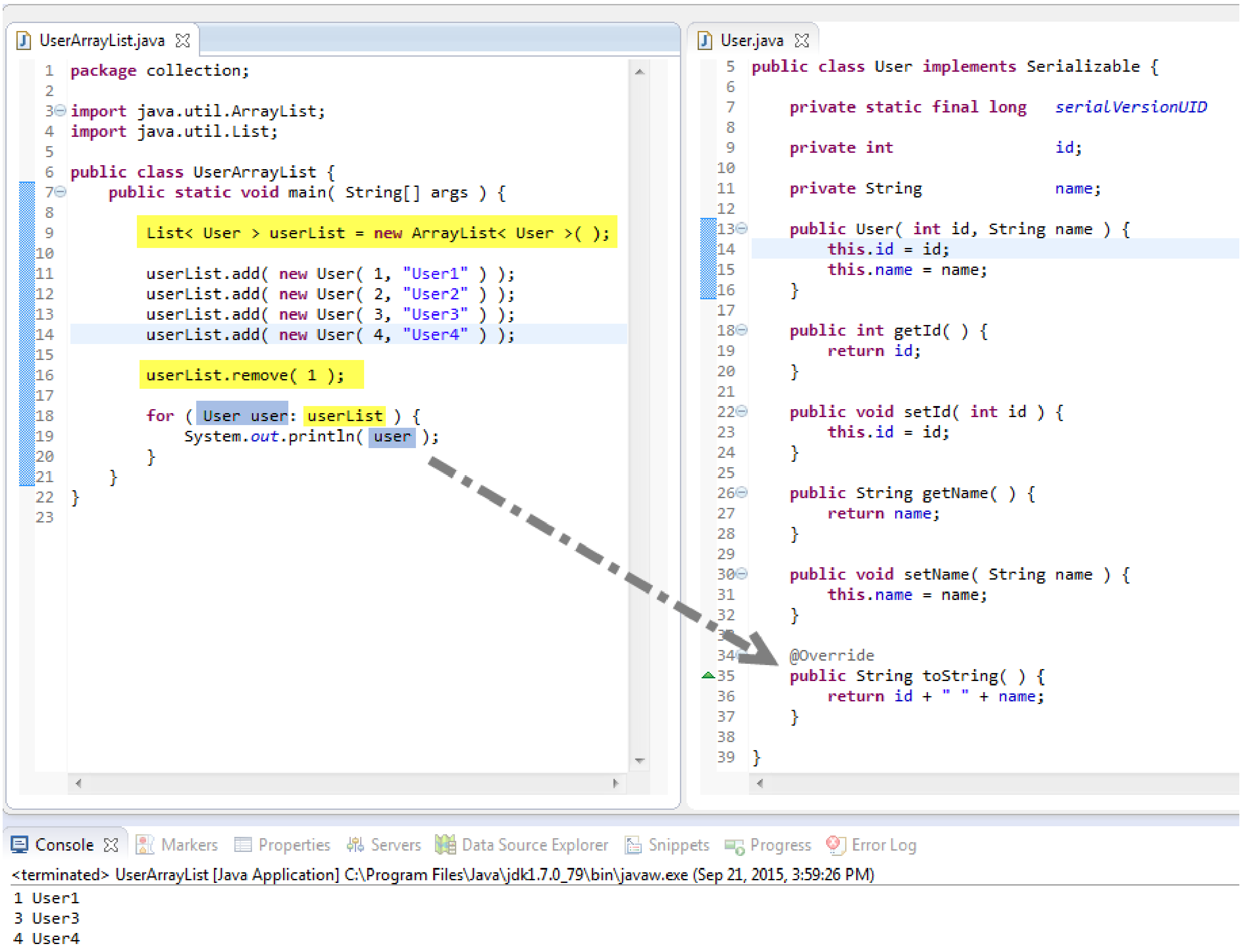


```java
package collection;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class ArrayListExample {
    public static void main( String[] args ) {

        List< String > countryList = new ArrayList< String >( );
        countryList.add( "Nepal" );
        countryList.add( "China" );
        countryList.add( "USA" );
        countryList.add( "Japan" );

        Iterator< String > itr = countryList.iterator( );// getting Iterator to traverse elements
        while ( itr.hasNext( ) ) {
            String country = itr.next( );
            System.out.println( country );
            if ( country.equals( "Japan" ) ) {
                itr.remove( );
            }
        }

        countryList.remove( 1 );

        System.out.println( "***********" );
        for ( String country: countryList ) {
            System.out.println( country );
        }

    }
}
```

<terminated> ArrayListExample [Java Application] C:\Program Files\Java\jdk1.7.0_79\bin\javaw.exe (Sep 21, 2015, 3:51:02 PM)

```
Nepal
China
USA
Japan
***********
Nepal
USA
```

User-defined class objects



```
package collection;

import java.util.ArrayList;
import java.util.List;

public class UserArrayList {
    public static void main( String[] args ) {

        List< User > userList = new ArrayList< User >( );

        userList.add( new User( 1, "User1" ) );
        userList.add( new User( 2, "User2" ) );
        userList.add( new User( 3, "User3" ) );
        userList.add( new User( 4, "User4" ) );

        userList.remove( 1 );

        for ( User user: userList ) {
            System.out.println( user );
        }
    }
}
```

```
public class User implements Serializable {

    private static final long   serialVersionUID

    private int                 id;

    private String              name;

    public User( int id, String name ) {
        this.id = id;
        this.name = name;
    }

    public int getId( ) {
        return id;
    }

    public void setId( int id ) {
        this.id = id;
    }

    public String getName( ) {
        return name;
    }

    public void setName( String name ) {
        this.name = name;
    }

    @Override
    public String toString( ) {
        return id + " " + name;
    }

}
```

```
<terminated> UserArrayList [Java Application] C:\Program Files\Java\jdk1.7.0_79\bin\javaw.exe (Sep 21, 2015, 3:59:26 PM)
1 User1
3 User3
4 User4
```

Java LinkedList class

- Java LinkedList class uses doubly linked list to store the elements. It extends the AbstractList class and implements List and Deque interfaces.

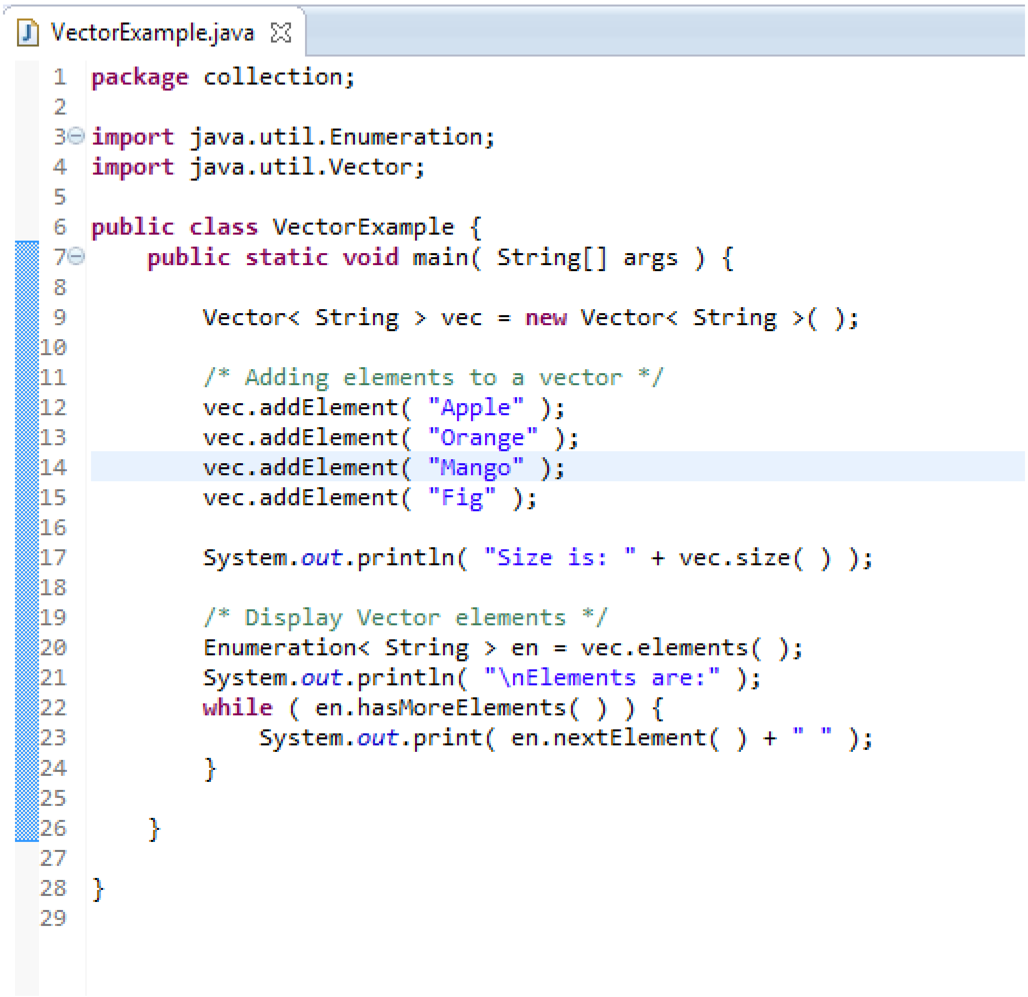
  LinkedList<String> al=**new** LinkedList<String>();

ArrayList	LinkedList
1) ArrayList internally uses **dynamic array** to store the elements.	LinkedList internally uses **doubly linked list** to store the elements.
2) Manipulation with ArrayList is **slow** because it internally uses array. If any element is removed from the array, all the bits are shifted in memory.	Manipulation with LinkedList is **faster** than ArrayList because it uses doubly linked list so no bit shifting is required in memory.

3) ArrayList class can **act as a list** only because it implements List only.	LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces.
4) ArrayList is **better for storing and accessing** data.	LinkedList is **better for manipulating** data.

Vector:

Vector is synchronized which means it is suitable for thread-safe operations but it *gives poor performance when used in* **multi-thread environment**. It is recommended to use ArrayList (it is non-synchronized, gives good performance) in place of Vector when there is no need of thread-safe operations.



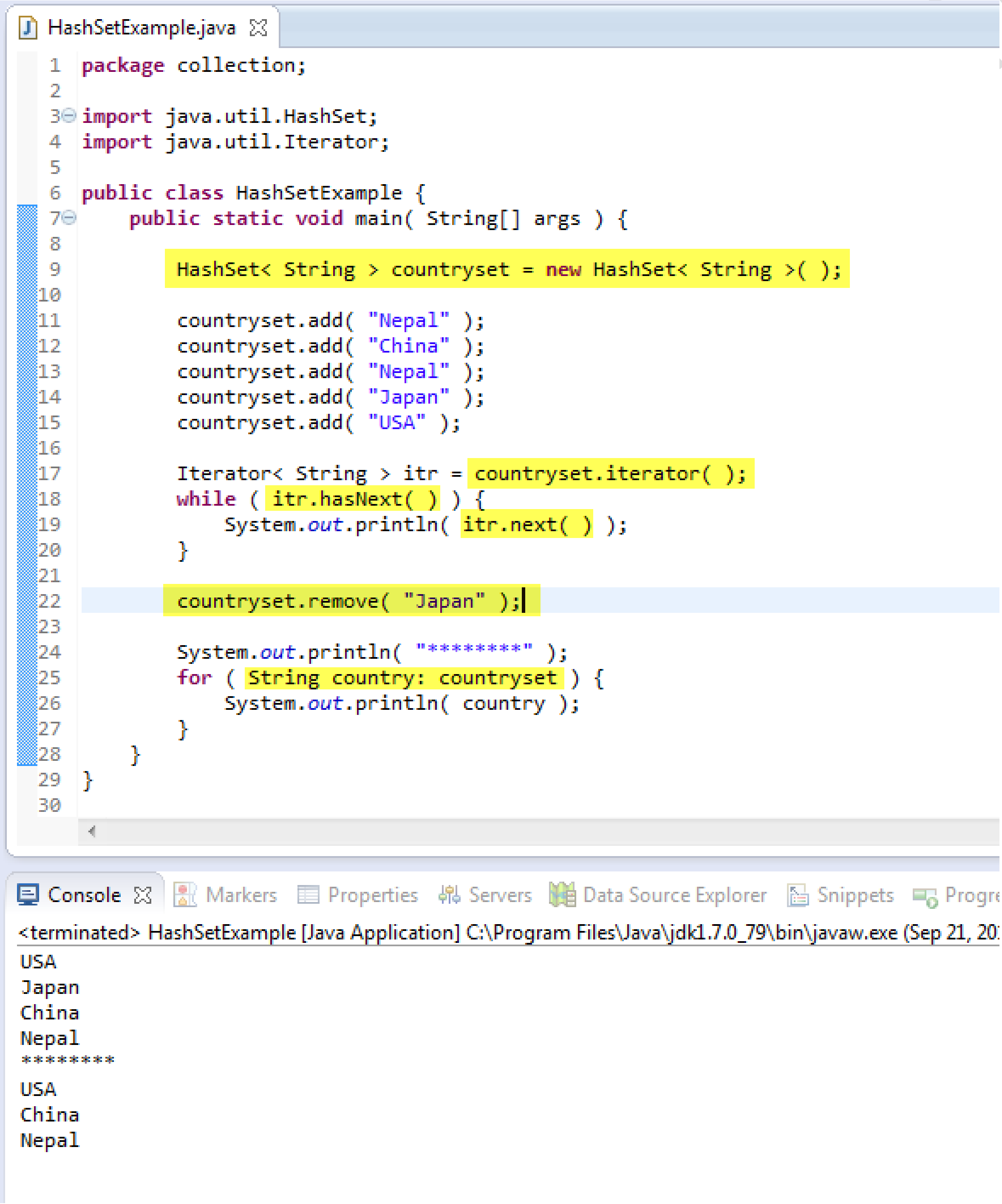
```java
package collection;

import java.util.Enumeration;
import java.util.Vector;

public class VectorExample {
    public static void main( String[] args ) {

        Vector< String > vec = new Vector< String >( );

        /* Adding elements to a vector */
        vec.addElement( "Apple" );
        vec.addElement( "Orange" );
        vec.addElement( "Mango" );
        vec.addElement( "Fig" );

        System.out.println( "Size is: " + vec.size( ) );

        /* Display Vector elements */
        Enumeration< String > en = vec.elements( );
        System.out.println( "\nElements are:" );
        while ( en.hasMoreElements( ) ) {
            System.out.print( en.nextElement( ) + " " );
        }

    }

}

```

Set (HashSet, LinkedHashSet, TreeSet)(Contain unique and sorted elements)

The Java platform contains three general-purpose Set implementations: `HashSet`, `TreeSet`, and `LinkedHashSet`. Set interface doesn't allow random-access to an element in the Collection. You can use iterator or foreach loop to traverse the elements of a Set.

Java HashSet class

- uses hashtable to store the elements. It extends AbstractSet class and implements Set interface.
- contains unique elements only.



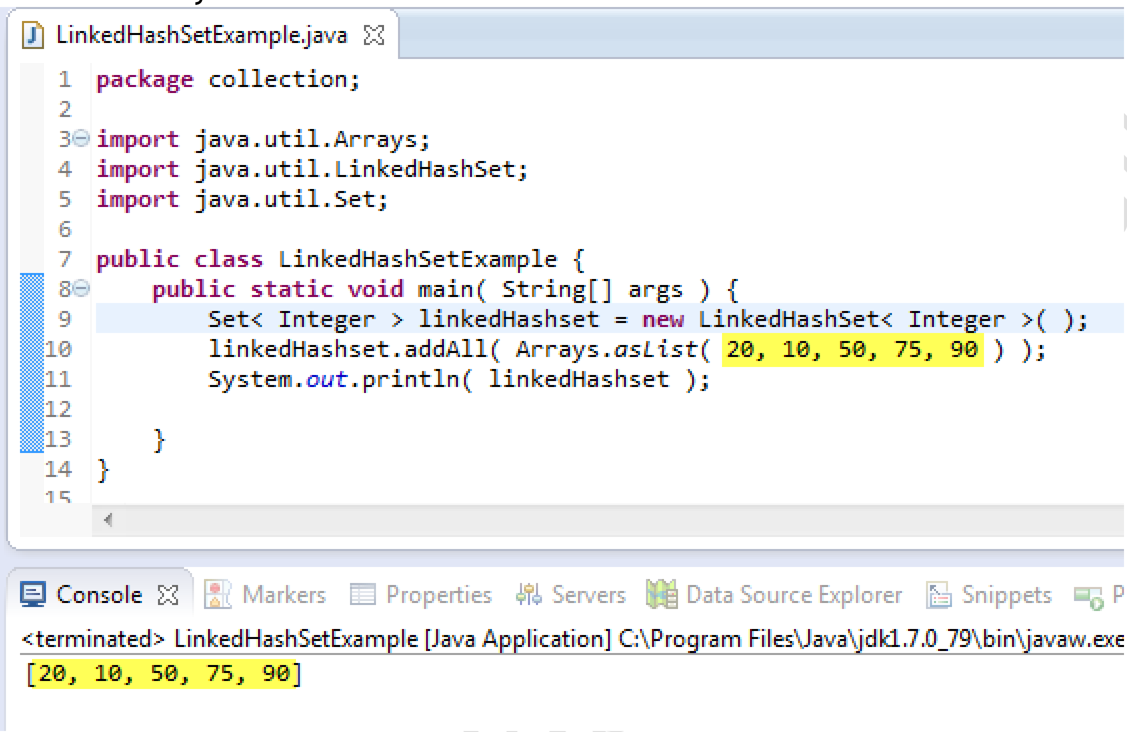
```java
package collection;

import java.util.HashSet;
import java.util.Iterator;

public class HashSetExample {
    public static void main( String[] args ) {

        HashSet< String > countryset = new HashSet< String >( );

        countryset.add( "Nepal" );
        countryset.add( "China" );
        countryset.add( "Nepal" );
        countryset.add( "Japan" );
        countryset.add( "USA" );

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

        countryset.remove( "Japan" );

        System.out.println( "********" );
        for ( String country: countryset ) {
            System.out.println( country );
        }
    }
}
```

```
<terminated> HashSetExample [Java Application] C:\Program Files\Java\jdk1.7.0_79\bin\javaw.exe (Sep 21, 20
USA
Japan
China
Nepal
********
USA
China
Nepal
```

LinkedHashSet:

LinkedHashSet is also an implementation of Set interface.

1. `HashSet` doesn’t maintain any kind of order of its elements.
2. `TreeSet` sorts the elements in ascending order.
3. `LinkedHashSet` maintains the insertion order. Elements gets sorted in the same sequence in which they have been added to the Set.

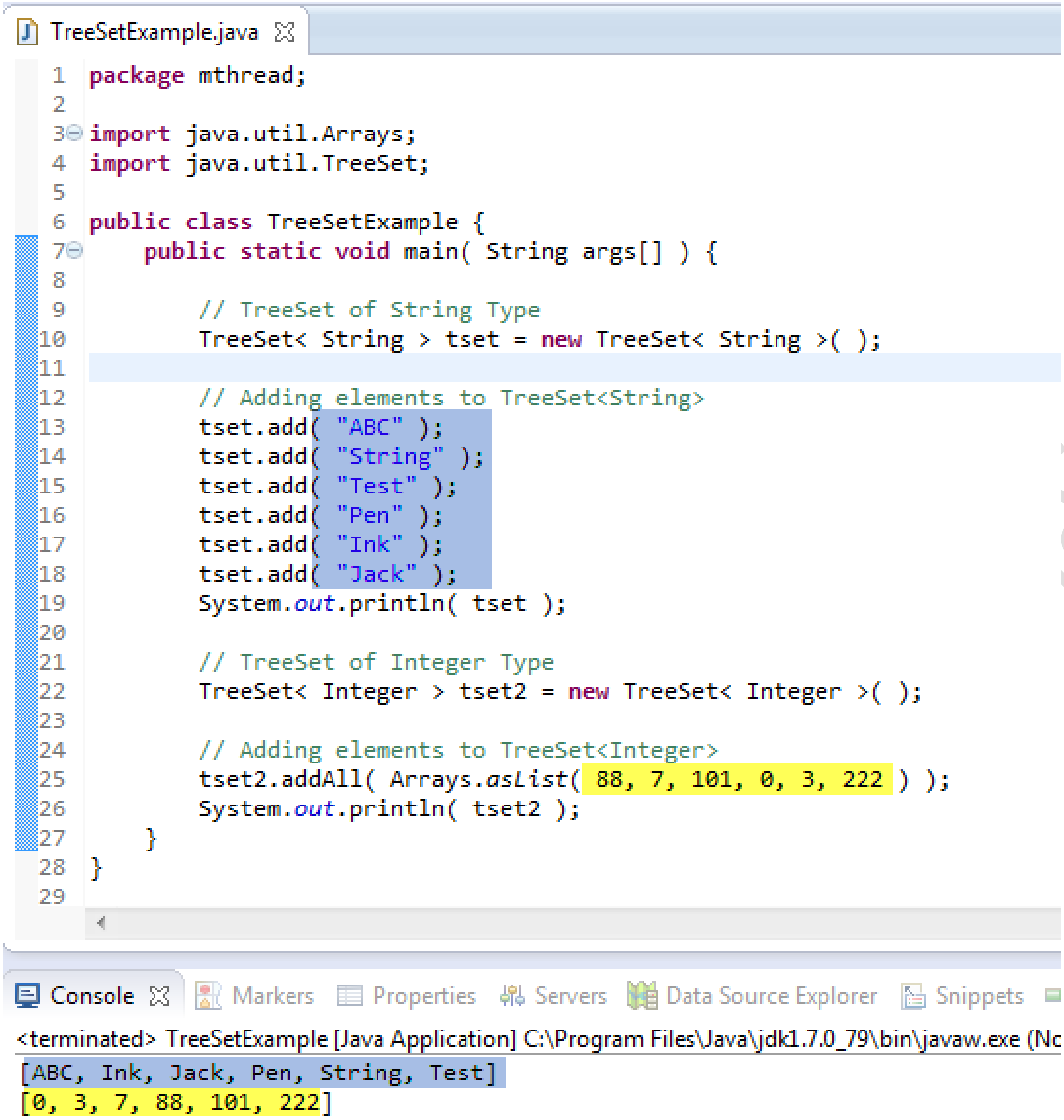


```
package collection;

import java.util.Arrays;
import java.util.LinkedHashSet;
import java.util.Set;

public class LinkedHashSetExample {
    public static void main( String[] args ) {
        Set< Integer > linkedHashset = new LinkedHashSet< Integer >( );
        linkedHashset.addAll( Arrays.asList( 20, 10, 50, 75, 90 ) );
        System.out.println( linkedHashset );

    }
}
```

```
<terminated> LinkedHashSetExample [Java Application] C:\Program Files\Java\jdk1.7.0_79\bin\javaw.exe
[20, 10, 50, 75, 90]
```

TreeSet

`TreeSet` is similar to HashSet except that it sorts the elements in the ascending order while HashSet doesn’t maintain any order. TreeSet allows null element but like HashSet it doesn’t allow. Like most of the other collection classes this class is also not synchronized, however it can be synchronized explicitly like this: `SortedSet s = Collections.synchronizedSortedSet(new TreeSet(...));`



```java
package mthread;

import java.util.Arrays;
import java.util.TreeSet;

public class TreeSetExample {
    public static void main( String args[] ) {

        // TreeSet of String Type
        TreeSet< String > tset = new TreeSet< String >( );

        // Adding elements to TreeSet<String>
        tset.add( "ABC" );
        tset.add( "String" );
        tset.add( "Test" );
        tset.add( "Pen" );
        tset.add( "Ink" );
        tset.add( "Jack" );
        System.out.println( tset );

        // TreeSet of Integer Type
        TreeSet< Integer > tset2 = new TreeSet< Integer >( );

        // Adding elements to TreeSet<Integer>
        tset2.addAll( Arrays.asList( 88, 7, 101, 0, 3, 222 ) );
        System.out.println( tset2 );
    }
}
```

```
<terminated> TreeSetExample [Java Application] C:\Program Files\Java\jdk1.7.0_79\bin\javaw.exe (Nc
[ABC, Ink, Jack, Pen, String, Test]
[0, 3, 7, 88, 101, 222]
```

Map

A Map is an object that maps keys to values. A map cannot contain duplicate keys. There are three main implementations of Map interfaces: HashMap, TreeMap, and LinkedHashMap.

HashMap: it makes no guarantees concerning the order of iteration

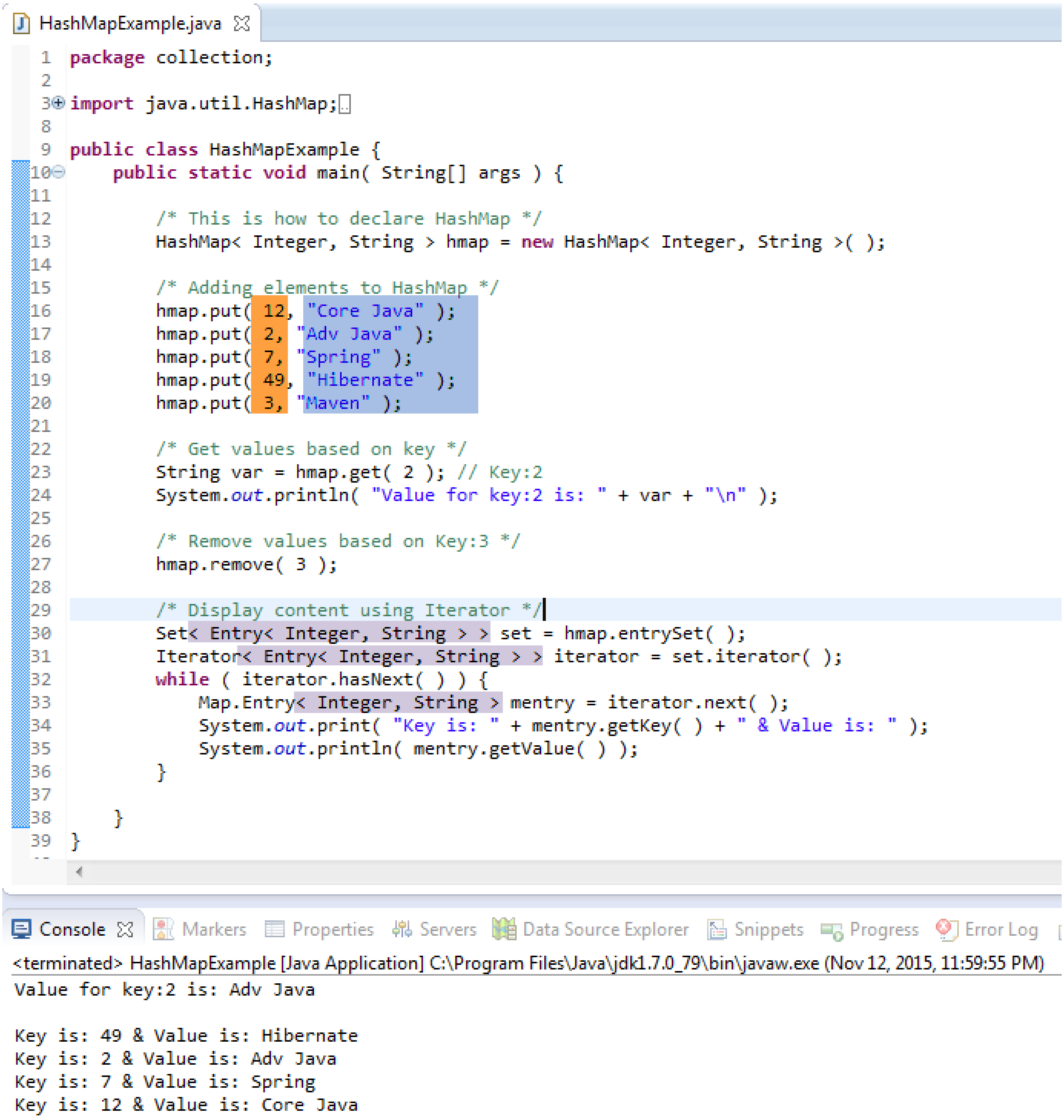
TreeMap: It stores its elements in a red-black tree, orders its elements based on their values; it is substantially slower than HashMap.

LinkedHashMap: It orders its elements based on the order in which they were inserted into the set (insertion-order).

- HashMap
- TreeMap
- LinkedHashMap

HashMap

HashMap is a Map based collection class that is used for storing Key & value pairs. This class makes no guarantees as to the order of the map. It is similar to the Hashtable class except that it is unsynchronized and permits nulls(null values and null key).

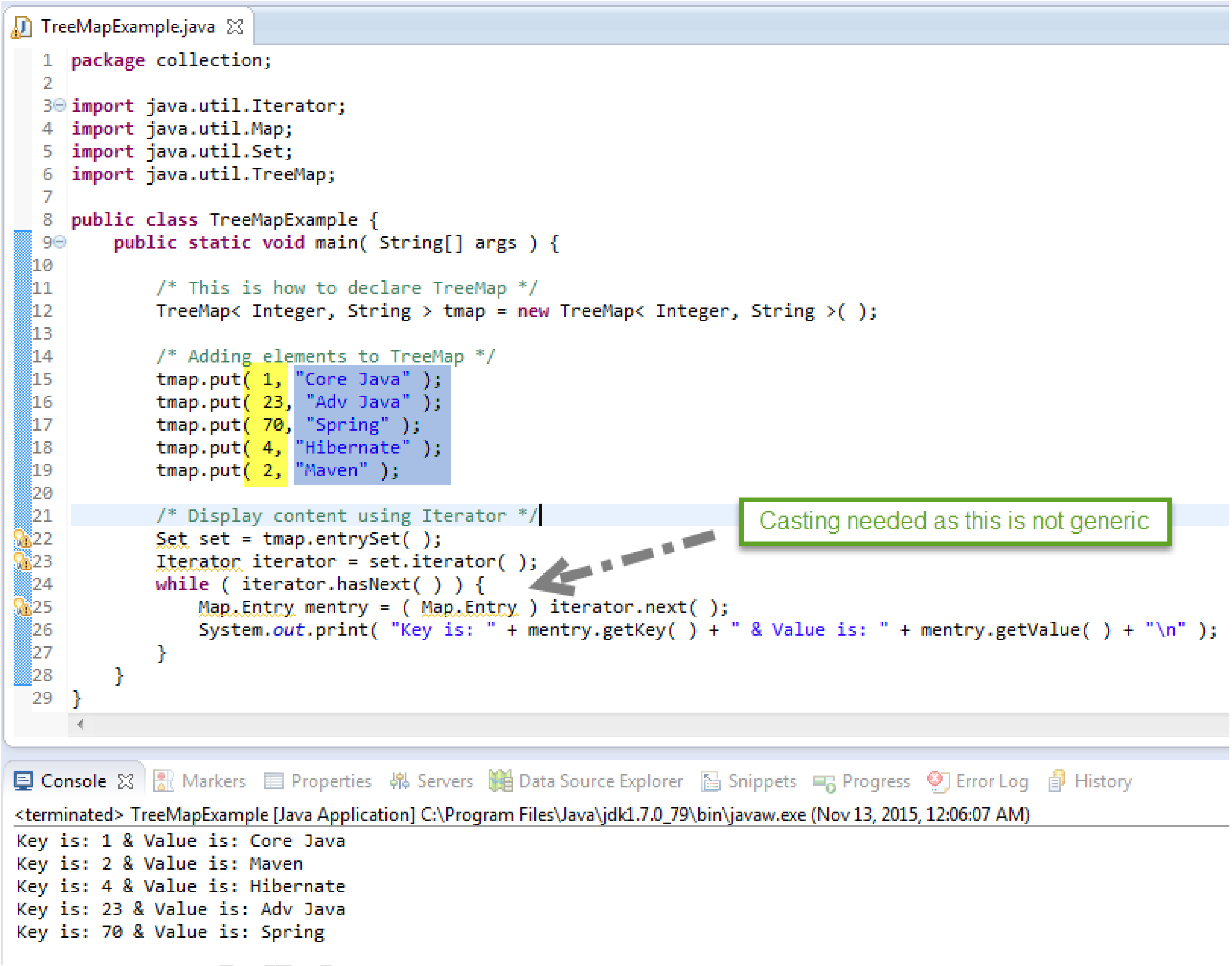


```java
package collection;

import java.util.HashMap;

public class HashMapExample {
    public static void main( String[] args ) {

        /* This is how to declare HashMap */
        HashMap< Integer, String > hmap = new HashMap< Integer, String >( );

        /* Adding elements to HashMap */
        hmap.put( 12, "Core Java" );
        hmap.put( 2, "Adv Java" );
        hmap.put( 7, "Spring" );
        hmap.put( 49, "Hibernate" );
        hmap.put( 3, "Maven" );

        /* Get values based on key */
        String var = hmap.get( 2 ); // Key:2
        System.out.println( "Value for key:2 is: " + var + "\n" );

        /* Remove values based on Key:3 */
        hmap.remove( 3 );

        /* Display content using Iterator */
        Set< Entry< Integer, String > > set = hmap.entrySet( );
        Iterator< Entry< Integer, String > > iterator = set.iterator( );
        while ( iterator.hasNext( ) ) {
            Map.Entry< Integer, String > mentry = iterator.next( );
            System.out.print( "Key is: " + mentry.getKey( ) + " & Value is: " );
            System.out.println( mentry.getValue( ) );
        }

    }
}
```

```
<terminated> HashMapExample [Java Application] C:\Program Files\Java\jdk1.7.0_79\bin\javaw.exe (Nov 12, 2015, 11:59:55 PM)
Value for key:2 is: Adv Java

Key is: 49 & Value is: Hibernate
Key is: 2 & Value is: Adv Java
Key is: 7 & Value is: Spring
Key is: 12 & Value is: Core Java
```

TreeMap

TreeMap is Red-Black tree based NavigableMap implementation. TreeMap is sorted in the ascending order of its keys. TreeMap is unsynchronized collection class which means it is not suitable for thread-safe operations until unless synchronized explicitly.



```java
package collection;

import java.util.Iterator;
import java.util.Map;
import java.util.Set;
import java.util.TreeMap;

public class TreeMapExample {
    public static void main( String[] args ) {

        /* This is how to declare TreeMap */
        TreeMap< Integer, String > tmap = new TreeMap< Integer, String >( );

        /* Adding elements to TreeMap */
        tmap.put( 1, "Core Java" );
        tmap.put( 23, "Adv Java" );
        tmap.put( 70, "Spring" );
        tmap.put( 4, "Hibernate" );
        tmap.put( 2, "Maven" );

        /* Display content using Iterator */
        Set set = tmap.entrySet( );
        Iterator iterator = set.iterator( );
        while ( iterator.hasNext( ) ) {
            Map.Entry mentry = ( Map.Entry ) iterator.next( );
            System.out.print( "Key is: " + mentry.getKey( ) + " & Value is: " + mentry.getValue( ) + "\n" );
        }
    }
}
```

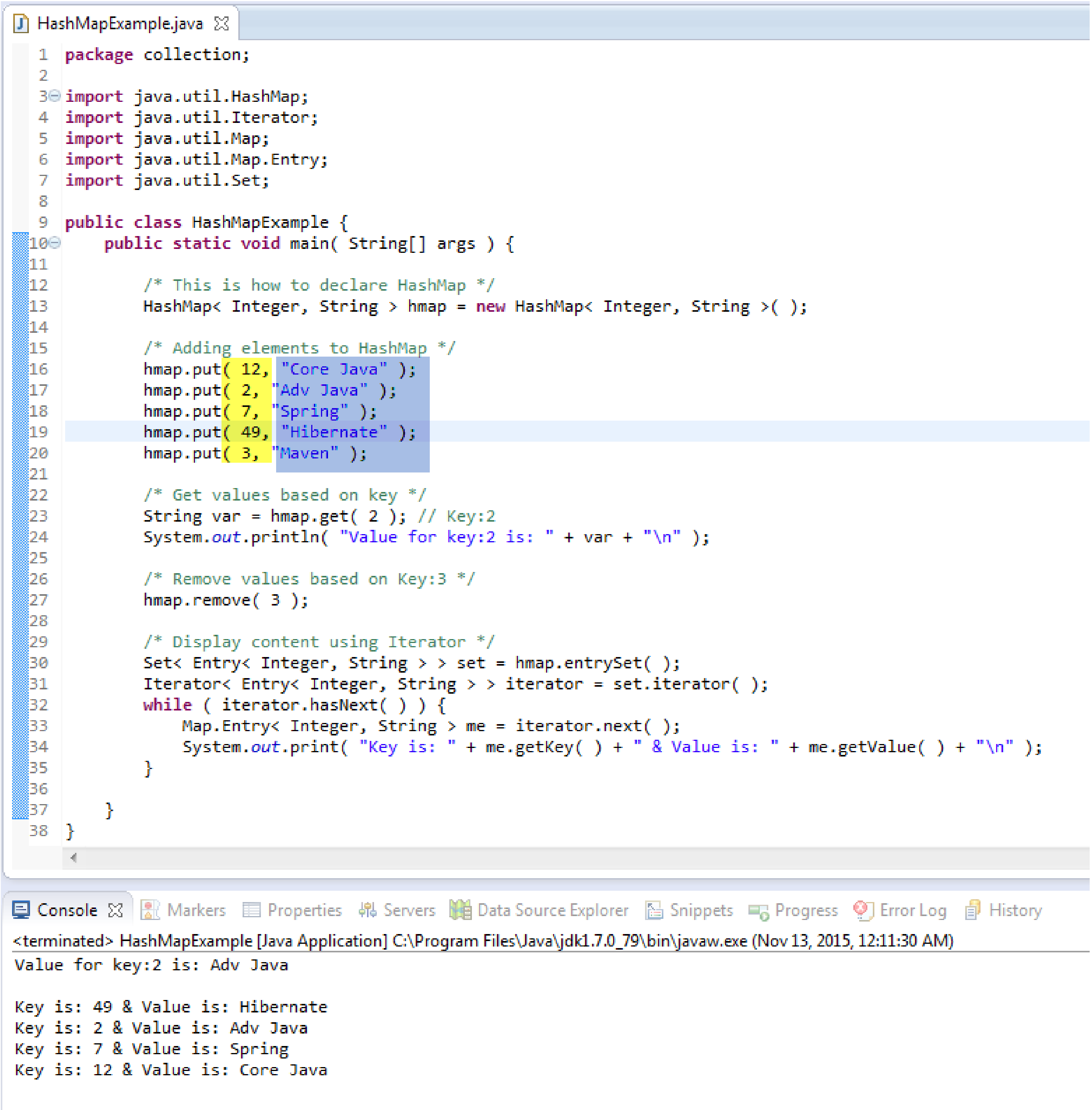
Casting needed as this is not generic

```
<terminated> TreeMapExample [Java Application] C:\Program Files\Java\jdk1.7.0_79\bin\javaw.exe (Nov 13, 2015, 12:06:07 AM)
Key is: 1 & Value is: Core Java
Key is: 2 & Value is: Maven
Key is: 4 & Value is: Hibernate
Key is: 23 & Value is: Adv Java
Key is: 70 & Value is: Spring
```

LinkedHashMap

LinkedHashMap is a Hash table and linked list implementation of the Map interface, with predictable iteration order.

- `HashMap` doesn’t maintain any order.
- `TreeMap` sort the entries in ascending order of keys.
- `LinkedHashMap` maintains the insertion order.



```java
package collection;

import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Set;

public class HashMapExample {
    public static void main( String[] args ) {

        /* This is how to declare HashMap */
        HashMap< Integer, String > hmap = new HashMap< Integer, String >( );

        /* Adding elements to HashMap */
        hmap.put( 12, "Core Java" );
        hmap.put( 2, "Adv Java" );
        hmap.put( 7, "Spring" );
        hmap.put( 49, "Hibernate" );
        hmap.put( 3, "Maven" );

        /* Get values based on key */
        String var = hmap.get( 2 ); // Key:2
        System.out.println( "Value for key:2 is: " + var + "\n" );

        /* Remove values based on Key:3 */
        hmap.remove( 3 );

        /* Display content using Iterator */
        Set< Entry< Integer, String > > set = hmap.entrySet( );
        Iterator< Entry< Integer, String > > iterator = set.iterator( );
        while ( iterator.hasNext( ) ) {
            Map.Entry< Integer, String > me = iterator.next( );
            System.out.print( "Key is: " + me.getKey( ) + " & Value is: " + me.getValue( ) + "\n" );
        }

    }
}
```

<terminated> HashMapExample [Java Application] C:\Program Files\Java\jdk1.7.0_79\bin\javaw.exe (Nov 13, 2015, 12:11:30 AM)

```
Value for key:2 is: Adv Java

Key is: 49 & Value is: Hibernate
Key is: 2 & Value is: Adv Java
Key is: 7 & Value is: Spring
Key is: 12 & Value is: Core Java
```

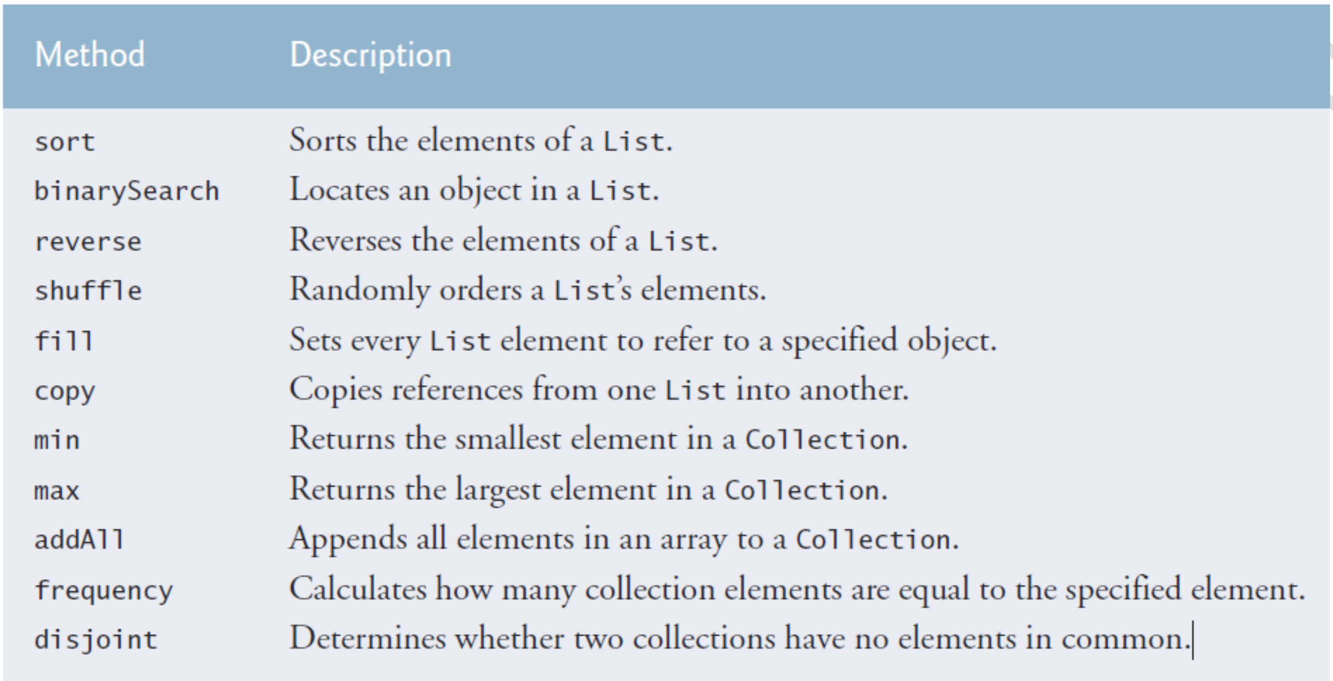
Iterator/ListIterator

Both Iterator and ListIterator are used to iterate through elements of a collection class. Using Iterator we can traverse in one direction (forward) while using ListIterator we can traverse the collection class on both the directions(backward and forward).

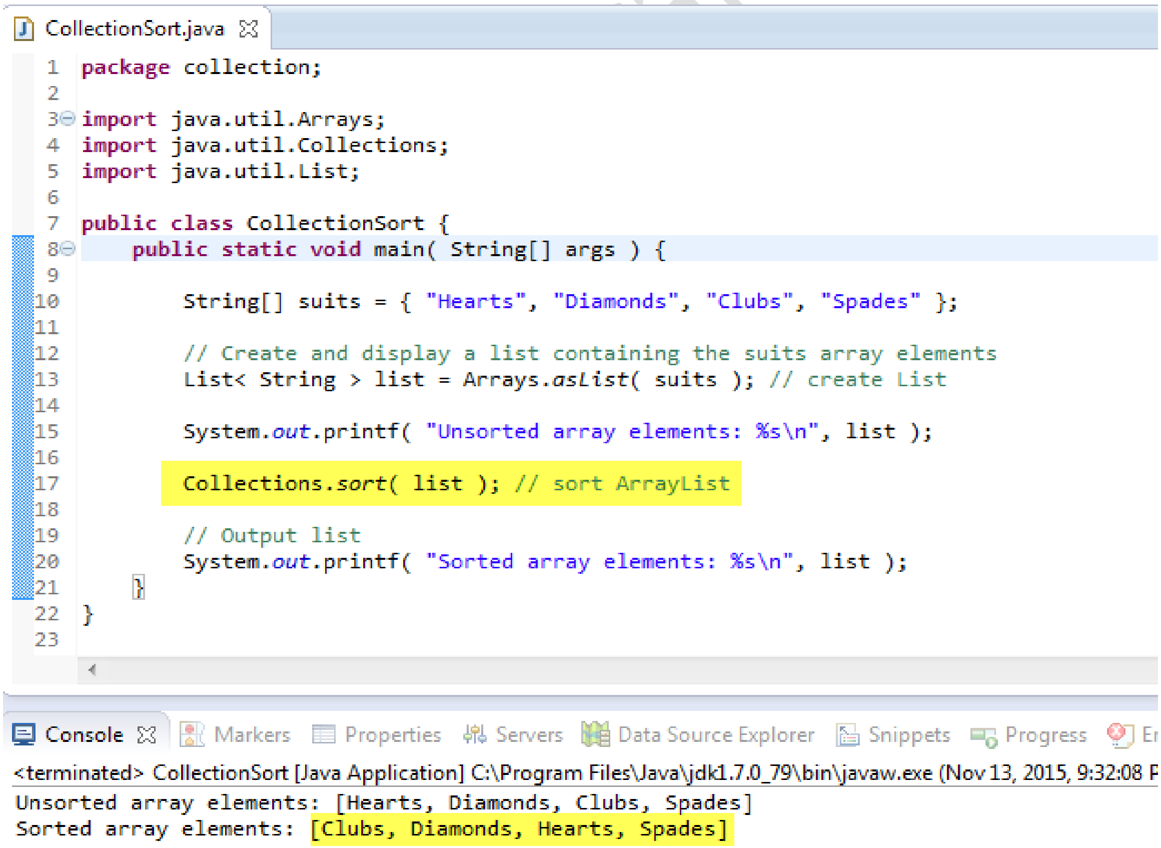
```
Methods: hasPrevious, previous, hasNext, next
```

Collections Methods:

Class Collections provides several high-performance algorithms for manipulating collection elements. The algorithms are implemented as static methods. The methods sort, binarySearch, reverse, shuffle, fill and copy operate on Lists. Methods min, max, addAll, frequency and disjoint operate on Collections.



Method	Description
`sort`	Sorts the elements of a `List`.
`binarySearch`	Locates an object in a `List`.
`reverse`	Reverses the elements of a `List`.
`shuffle`	Randomly orders a `List`'s elements.
`fill`	Sets every `List` element to refer to a specified object.
`copy`	Copies references from one `List` into another.
`min`	Returns the smallest element in a `Collection`.
`max`	Returns the largest element in a `Collection`.
`addAll`	Appends all elements in an array to a `Collection`.
`frequency`	Calculates how many collection elements are equal to the specified element.
`disjoint`	Determines whether two collections have no elements in common.

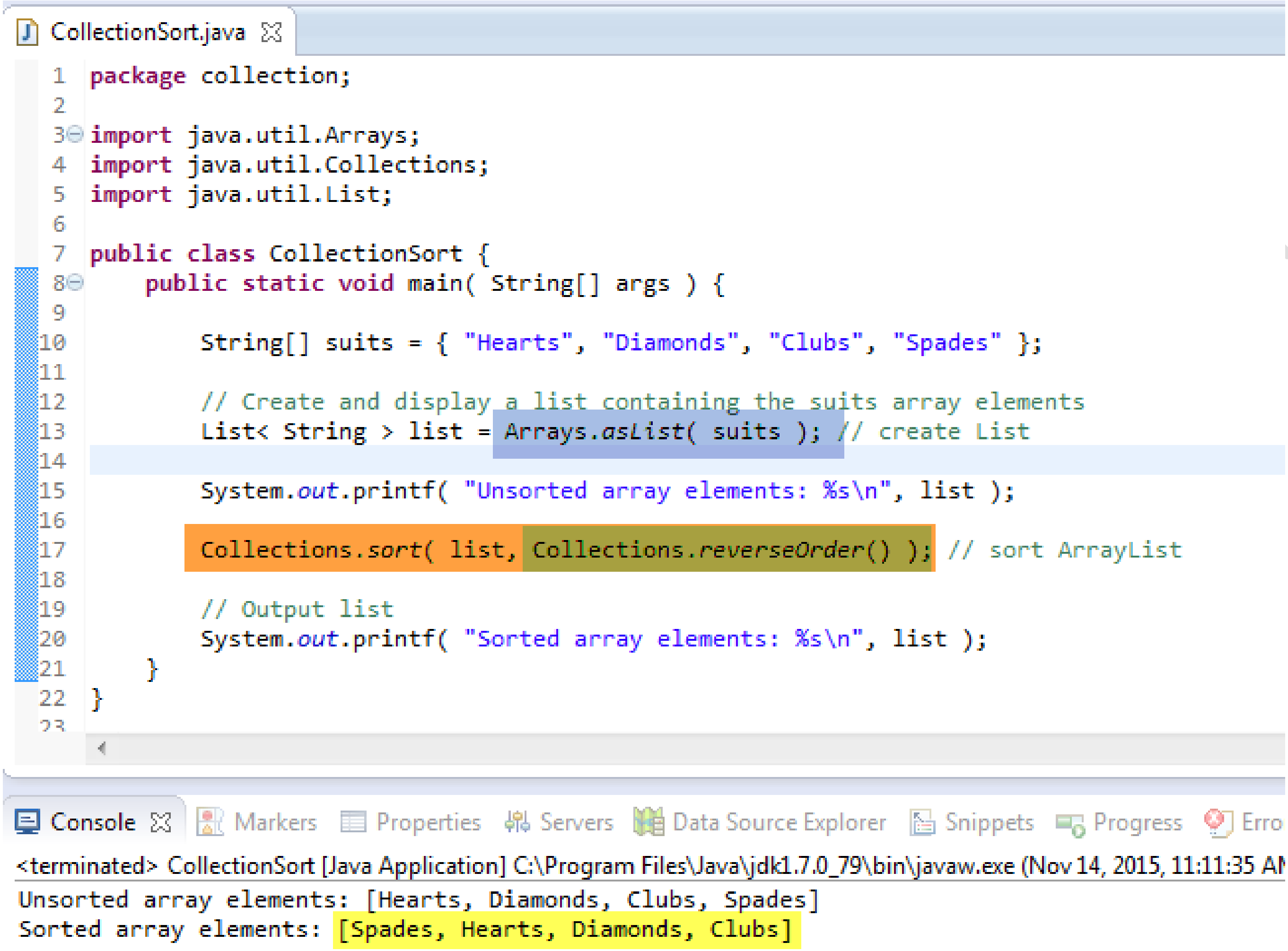


```java
package collection;

import java.util.Arrays;
import java.util.Collections;
import java.util.List;

public class CollectionSort {
    public static void main( String[] args ) {

        String[] suits = { "Hearts", "Diamonds", "Clubs", "Spades" };

        // Create and display a list containing the suits array elements
        List< String > list = Arrays.asList( suits ); // create List

        System.out.printf( "Unsorted array elements: %s\n", list );

        Collections.sort( list ); // sort ArrayList

        // Output list
        System.out.printf( "Sorted array elements: %s\n", list );
    }
}

```

<terminated> CollectionSort [Java Application] C:\Program Files\Java\jdk1.7.0_79\bin\javaw.exe (Nov 13, 2015, 9:32:08 P

```
Unsorted array elements: [Hearts, Diamonds, Clubs, Spades]
Sorted array elements: [Clubs, Diamonds, Hearts, Spades]
```

Sorting in Descending Order



```java
package collection;

import java.util.Arrays;
import java.util.Collections;
import java.util.List;

public class CollectionSort {
    public static void main( String[] args ) {

        String[] suits = { "Hearts", "Diamonds", "Clubs", "Spades" };

        // Create and display a list containing the suits array elements
        List< String > list = Arrays.asList( suits ); // create List

        System.out.printf( "Unsorted array elements: %s\n", list );

        Collections.sort( list, Collections.reverseOrder() ); // sort ArrayList

        // Output list
        System.out.printf( "Sorted array elements: %s\n", list );
    }
}
```

```
<terminated> CollectionSort [Java Application] C:\Program Files\Java\jdk1.7.0_79\bin\javaw.exe (Nov 14, 2015, 11:11:35 AI
Unsorted array elements: [Hearts, Diamonds, Clubs, Spades]
Sorted array elements: [Spades, Hearts, Diamonds, Clubs]
```