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

public class Main {
  public static void main(String[] args) {
    Map errors = new HashMap();

    errors.put("404", "A.");
    errors.put("403", "B.");
    errors.put("500", "C.");

    String errorDesc = (String) errors.get("404");
    System.out.println("Error 404: " + errorDesc);

    Iterator iterator = errors.keySet().iterator();
    while (iterator.hasNext()) {
      String key = (String) iterator.next();
      System.out.println("Error " + key + " means " + errors.get(key));
    }
  }
}
```

```java
import java.util.HashMap;

public class MainClass {
  public static void main(String[] a) {

    HashMap map = new HashMap();
    map.put("key1", "value1");
    map.put("key2", "value2");
    map.put("key3", "value3");

    HashMap map2 = (HashMap)map.clone();
    System.out.println(map2);
  }
}
```

```
{key1=value1, key3=value3, key2=value2}
```

Create Java Hashtable from HashMap

```java
import java.util.Enumeration;
import java.util.HashMap;
import java.util.Hashtable;

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

    HashMap<String, String> hMap = new HashMap<String, String>();

    hMap.put("1", "One");
    hMap.put("2", "Two");
    hMap.put("3", "Three");

    Hashtable<String, String> ht = new Hashtable<String, String>();
    ht.put("1", "REPLACED !!");
    ht.put("4", "Four");

    Enumeration e = ht.elements();
    while (e.hasMoreElements()){
      System.out.println(e.nextElement());
    }

    ht.putAll(hMap);
    e = ht.elements();

    while (e.hasMoreElements()){
      System.out.println(e.nextElement());
    }
  }
}
```

Java - The Hashtable Class

```
import java.util.Enumeration;
import java.util.HashMap;
import java.util.Hashtable;

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

    HashMap<String, String> hMap = new HashMap<String, String>();

    hMap.put("1", "One");
    hMap.put("2", "Two");
    hMap.put("3", "Three");

    Hashtable<String, String> ht = new Hashtable<String, String>();
    ht.put("1", "REPLACED !!");
    ht.put("4", "Four");

    Enumeration e = ht.elements();
    while (e.hasMoreElements()){
      System.out.println(e.nextElement());
    }

    ht.putAll(hMap);
    e = ht.elements();

    while (e.hasMoreElements()){
      System.out.println(e.nextElement());
    }
  }
}
```

`Hashtable( )`

`Hashtable(Map m)`

`Hashtable(int size)`

void clear( )
Resets and empties the hash table.

boolean contains(Object value)
Returns true if some value equal to value exists within the hash table. Returns false if the value isn't found.

boolean containsKey(Object key)
Returns true if some key equal to key exists within the hash table. Returns false if the key isn't found.

boolean containsValue(Object value)
Returns true if some value equal to value exists within the hash table. Returns false if the value isn't found.

Enumeration elements( )
Returns an enumeration of the values contained in the hash table.

Object get(Object key)
Returns the object that contains the value associated with key. If key is not in the hash table, a null object is returned.

boolean isEmpty( )
Returns true if the hash table is empty; returns false if it contains at least one key.

Enumeration keys( )
Returns an enumeration of the keys contained in the hash table.

Object put(Object key, Object value)
Inserts a key and a value into the hash table. Returns null if key isn't already in the hash table; returns the previous value associated with key if key is already in the hash table.

void rehash( )
Increases the size of the hash table and rehashes all of its keys.

Object remove(Object key)
Removes key and its value. Returns the value associated with key. If key is not in the hash table, a null object is returned.

int size( )
Returns the number of entries in the hash table.

String toString( )
Returns the string equivalent of a hash table.

```
import java.util.Arrays;
import java.util.HashSet;
import java.util.Set;

public class MainClass {

  public static void main(String[] a) {
    String elements[] = { "A", "B", "C", "D", "E" };
    Set set = new HashSet(Arrays.asList(elements));

    elements = new String[] { "E", "F" };

    set.addAll(Arrays.asList(elements));

    System.out.println(set);
  }
}
```

```
[D, A, F, C, B, E]
```

If not found, false is returned.

If removal is not supported, you'll get an UnsupportedOperationException thrown.

```
import java.util.Arrays;
import java.util.HashSet;
import java.util.Set;

public class MainClass {

  public static void main(String[] a) {
    String elements[] = { "A", "B", "C", "D", "E" };
    Set set = new HashSet(Arrays.asList(elements));

    set.remove("A");

    System.out.println(set);
  }
}
```

```
[D, C, B, E]
```

» HashSet

```
import java.util.Arrays;
import java.util.HashSet;
import java.util.Set;

public class MainClass {

  public static void main(String[] a) {
    String elements[] = { "A", "B", "C", "D", "E" };
    Set set = new HashSet(Arrays.asList(elements));

    elements = new String[] { "E", "F" };

    set.addAll(Arrays.asList(elements));

    System.out.println(set);

    set.clear();

    System.out.println(set);
  }
}
```

```
[D, A, F, C, B, E]
 []
```

```java
import java.util.Vector;

public class MainClass {
  public static void main(String args[]) {
    Vector v = new Vector(5);
    for (int i = 0; i < 10; i++) {
      v.add(0,i);
    }
    System.out.println(v);

    Vector v2 = new Vector(5);
    for (int i = 0; i < 10; i++) {
      v2.add(0,i);
    }
    System.out.println(v2);

    System.out.println(v2.equals(v));

    v2.removeAll(v);

    System.out.println(v2);
  }

}
```

```
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
true
[]
```

Retaining Another Collection: public boolean retainAll(Collection c)

```java
import java.util.Vector;

public class MainClass {
  public static void main(String args[]) {
    Vector v = new Vector(5);
    for (int i = 0; i < 10; i++) {
      v.add(0,i);
    }
    System.out.println(v);

    Vector v2 = new Vector(5);
    for (int i = 0; i < 5; i++) {
      v2.add(0,i);
    }
    System.out.println(v2);
    System.out.println(v2.equals(v));

    v.retainAll(v2);
    System.out.println(v);
  }
}
```

intersection

```
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
[4, 3, 2, 1, 0]
false
[4, 3, 2, 1, 0]
```

```
public Object set(int index, Object element)
public void setElementAt(Object obj, int index)
```

```
import java.util.Vector;

public class MainClass {
  public static void main(String args[]) {
    Vector v = new Vector(5);
    for (int i = 0; i < 10; i++) {
      v.add(0,i);
    }
    System.out.println(v);

    for (int i = 0; i < 10; i++) {
      v.set(i,"A");
    }

    System.out.println(v);
  }
}
```

```
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
[A, A, A, A, A, A, A, A, A, A]
```

```
import java.util.Vector;

public class Main {

  public static void main(String[] args) {
    Vector<String> v = new Vector<String>();
    v.add("1");
    v.add("2");
    v.add("3");
    v.add("4");
    v.add("5");
    v.add("1");
    v.add("2");

    System.out.println(v.contains("3"));
    System.out.println(v.indexOf("5"));
    System.out.println(v.lastIndexOf("2"));
  }
}
```

Vector Immutability

```
import java.util.Collections;
import java.util.List;
import java.util.Vector;

public class MainClass {
  public static void main(String args[]) {
    Vector v = new Vector(5);
    for (int i = 0; i < 10; i++) {
      v.add(0, i);
    }
    System.out.println(v);
    System.out.println(v.size());

    List l = Collections.unmodifiableList(v);

    l.add(1);
    System.out.println(l);
  }
}
```

```
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
Exception in thread "main" java.lang.UnsupportedOperationException
        at java.util.Collections$UnmodifiableCollection.add(Collections.java:101
        at MainClass.main(MainClass.java:16)
```

```
public Object get(int index)
public Object elementAt(int index)
(starting from zero)
```

```
public Object firstElement()
public Object lastElement()
```

```
import java.util.Vector;

public class MainClass {
  public static void main(String args[]) {
    Vector v = new Vector();
    v.add("Hello");
    String s = (String)v.get(0);
    System.out.println(s);
  }
}
```

```
Hello
```

```
import java.util.Vector;

public class MainClass {
  public static void main(String args[]) {
    Vector<String> v = new Vector<String>();
    v.add("A");
    v.add("B");
    String firstElement = (String) v.firstElement();
    String lastElement = (String) v.lastElement();

    System.out.println(firstElement);
    System.out.println(lastElement);
  }
}
```

```
A
B
```

```
import java.util.Iterator;
import java.util.Vector;

public class MainClass {
  public static void main(String args[]) {
    Vector<String> v = new Vector<String>();
    v.add("A");
    v.add("B");

    Iterator i = v.iterator();
    while (i.hasNext()) {
      System.out.println(i.next());
    }
  }
}
```

```
A
B
```

```
import java.util.Vector;

public class MainClass {
  public static void main(String args[]) {
    Vector v = new Vector();
    v.add("A");
    v.add("B");

    boolean isContaining = v.contains("B");

    System.out.println(isContaining);
  }

}
```

```
true
```

```
public int lastIndexOf(Object element)
public int lastIndexOf(Object element, int index)
```

These methods use equals() to check for equality.

Return -1 if not found.

The lastIndexOf() reports the position from the beginning, but starts at the end.

```
import java.util.Vector;

public class MainClass {
  static String members[] = { "A", "I", "C", "D", "E", "F", "G", "H", "I", "J" };

  public static void main(String args[]) {
    Vector v = new Vector();
    for (int i = 0, n = members.length; i < n; i++) {
      v.add(members[i]);
    }
    System.out.println(v);
    System.out.println(v.lastIndexOf("I"));
  }
}
```

```
[A, I, C, D, E, F, G, H, I, J]
8
```

Finding all the positions for a single element

```
import java.util.Vector;

public class MainClass {

  public static void main(String args[]) {
    String members[] = { "A", "I", "C", "D", "E", "F", "G", "H", "I", "J" };
    Object value = "I";
    Vector v = new Vector();
    for (int i = 0, n = members.length; i < n; i++) {
      v.add(members[i]);
    }
    int index = 0;
    int length = v.size();
    while ((index < length) && (index >= 0)) {
      index = v.indexOf(value, index);
      if (index != -1) {
        System.out.println(index);
        index++;
      }
    }
  }
}
```

```
1
8
```

- **public int indexOf(int ch):** Returns the index within this string of the first occurrence of the specified character or -1 if the character does not occur.
- **public int indexOf(int ch, int fromIndex):** Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index or -1 if the character does not occur.
- **int indexOf(String str):** Returns the index within this string of the first occurrence of the specified substring. If it does not occur as a substring, -1 is returned.
- **int indexOf(String str, int fromIndex):** Returns the index within this string of the first occurrence of the specified substring, starting at the specified index. If it does not occur, -1 is returned.

Try it

```
import java.io.*;

public class Test {

   public static void main(String args[]) {
      String Str = new String("Welcome to Tutorialspoint.com");
      String SubStr1 = new String("Tutorials");
      String SubStr2 = new String("Sutorials");

      System.out.print("Found Index :" );
      System.out.println(Str.indexOf( 'o' ));
      System.out.print("Found Index :" );
      System.out.println(Str.indexOf( 'o', 5 ));
      System.out.print("Found Index :" );
      System.out.println( Str.indexOf( SubStr1 ));
      System.out.print("Found Index :" );
      System.out.println( Str.indexOf( SubStr1, 15 ));
      System.out.print("Found Index :" );
      System.out.println(Str.indexOf( SubStr2 ));
   }
}
```

This produces the following result:

```
Found Index :4
Found Index :9
Found Index :11
Found Index :-1
Found Index :-1
```

Check if 2 strings are rotated versions of each other

In Java:

```
boolean isRotation(String s1,String s2) {
    return (s1!=null && s2!=null &&
            s1.length() == s2.length()) && ((s1+s1).indexOf(s2) != -1);
}
```

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

public boolean areAnagrams(String s1, String s2) {
    char[] ch1 = s1.toCharArray();
    char[] ch2 = s2.toCharArray();
    Arrays.sort(ch1);
    Arrays.sort(ch2);
    return Arrays.equals(ch1,ch2);
}
```

Java - The WeakHashMap Class

WeakHashMap is an implementation of the Map interface that stores only weak references to its keys. Storing only weak references allows a key-value pair to be garbagecollected when its key is no longer referenced outside of the WeakHashMap.

This class provides the easiest way to harness the power of weak references. It is useful for implementing "registry-like" data structures, where the utility of an entry vanishes when its key is no longer reachable by any thread.

The WeakHashMap functions identically to the HashMap with one very important exception: if the Java memory manager no longer has a strong reference to the object specified as a key, then the entry in the map will be removed.

Weak Reference: If the only references to an object are weak references, the garbage collector can reclaim the object's memory at any time.it doesn't have to wait until the system runs out of memory. Usually, it will be freed the next time the garbage collector runs.

The WeakHashMap class supports four constructors. The first form constructs a new, empty WeakHashMap with the default initial capacity (16) and the default load factor (0.75):

```
WeakHashMap()
```

The second form constructs a new, empty WeakHashMap with the given initial capacity and the default load factor, which is 0.75:

```
WeakHashMap(int initialCapacity)
```

The third form constructs a new, empty WeakHashMap with the given initial capacity and the given load factor.

```
WeakHashMap(int initialCapacity, float loadFactor)
```

The fourth form constructs a new WeakHashMap with the same mappings as the specified Map:

```
WeakHashMap(Map t)
```

1	**void clear()** Removes all mappings from this map.
2	**boolean containsKey(Object key)** Returns true if this map contains a mapping for the specified key.
3	**boolean containsValue(Object value)** Returns true if this map maps one or more keys to the specified value.
4	**Set entrySet()** Returns a collection view of the mappings contained in this map.
5	**Object get(Object key)** Returns the value to which the specified key is mapped in this weak hash map, or null if the map contains no mapping for this key.
6	**boolean isEmpty()** Returns true if this map contains no key-value mappings.
7	**Set keySet()** Returns a set view of the keys contained in this map.
8	**Object put(Object key, Object value)** Associates the specified value with the specified key in this map.
9	**void putAll(Map m)** Copies all of the mappings from the specified map to this map These mappings will replace any mappings that this map had for any of the keys currently in the specified map.
10	**Object remove(Object key)** Removes the mapping for this key from this map if present.
11	**int size()** Returns the number of key-value mappings in this map.
12	**Collection values()** Returns a collection view of the values contained in this map.

```
import java.util.*;

public class WeakHashMap {
   private static Map map;
   public static void main (String args[]) {
      map = new WeakHashMap();
      map.put(new String("Maine"), "Augusta");

      Runnable runner = new Runnable() {
         public void run() {
            while (map.containsKey("Maine")) {
               try {
                  Thread.sleep(500);
               } catch (InterruptedException ignored) {
               }
               System.out.println("Thread waiting");
               System.gc();
            }
         }
      };
      Thread t = new Thread(runner);
      t.start();
      System.out.println("Main waiting");
      try {
         t.join();
      } catch (InterruptedException ignored) {
      }
   }
}
```

This would produce the following result:

```
Main waiting
Thread waiting
```

If you do not include the call to System.gc(), the system may never run the garbage collector as no much memory is used by the program. For a more active program, the call would be unnecessary.