LA CLASE VECTOR

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

/**
 * Vector es similar al ArrayList, pero está sincronizada
 * (significa que un método no puede ser ejecutado por más
 * de un hilo (thread) a la vez).
 * Vector duplica su espacio cada vez que se añade elementos.
 * Se puede acceder a los elementos directamente mediante los
 * métodos get y set.
 *
 *   add es O(1)
 *   remove es O(n)
 *   set es O(1)
 *   get es O(1)
 *
 */
public class EjemploVector {
	@SuppressWarnings("unchecked")
	public static void main(String[] args) {
		Vector<Character> v = new Vector<Character>();

		/* ¿El arreglo esta vacío? */
		System.out.println(v.isEmpty());			// true

		v.add('a');						// ['a']
		v.add('b');						// ['a', 'b']
		v.add('c');						// ['a', 'b', 'c']
		System.out.println(v.size());			// 3
		System.out.println(v.get(1));			// b

		System.out.println(v);				// [a, b, c]
		System.out.println(v.contains('b'));		// true
		System.out.println(v.indexOf('b'));		// 1
		System.out.println(v.lastIndexOf('b'));		// 1
		v.set(2, 'x');					// ['a', 'b', 'x']
		v.remove(1);					// ['a', 'x']

		/* Con iterador */
		Iterator<Character> it = v.iterator();
		System.out.println(it.hasNext());		// true
		System.out.println(it.next());			// a
		it.remove();					// ['x']

		/* Saca una copia */
		Vector<Character> copia = (Vector<Character>) v.clone();

		System.out.println(copia);			// [x]
		v.clear();					// Elimina todos los elementos.
		System.out.println(v.isEmpty());		// true
	}
}
```

LA CLASE ARRAYLIST

```java
import java.util.ArrayList;
import java.util.Iterator;

/**
 * ArrayList se implementa como un arreglo de tamaño variable.
 * ArrayList crece el 50% cada vez que se añade elementos,
.* Se puede acceder directamente a los elementos mediante los
 * métodos get y set.
 *
 *   add es O(1)
 *   remove es O(n)
 *   set es O(1)
 *   get es O(1)
 *
 */
public class EjemploArrayList {
	@SuppressWarnings("unchecked")
	public static void main(String[] args) {
		ArrayList<Character> a = new ArrayList<Character>();

		/* ¿El arreglo esta vacío? */
		System.out.println(a.isEmpty());			// true

		a.add('a');						// ['a']
		a.add('b');						// ['a', 'b']
		a.add('c');						// ['a', 'b', 'c']
		System.out.println(a.size());				// 3
		System.out.println(a.get(1));				// b

		System.out.println(a);					// [a, b, c]
		System.out.println(a.contains('b'));			// true
		System.out.println(a.indexOf('b'));			// 1
		System.out.println(a.lastIndexOf('b'));			// 1
		a.set(2, 'x');						// ['a', 'b', 'x']
		a.remove(1);						// ['a', 'x']

		/* Con iterador */
		Iterator<Character> it = a.iterator();
		System.out.println(it.hasNext());			// true
		System.out.println(it.next());				// a
		it.remove();						// ['x']

		/* Saca una copia */
		ArrayList<Character> copia = (ArrayList<Character>) a.clone();

		System.out.println(copia);				// [x]
		a.clear();						// Elimina todos los elementos.
		System.out.println(a.isEmpty());			// true
	}
}
```

LA CLASE LINKEDLIST

```java
import java.util.LinkedList;
import java.util.ListIterator;
/**
 * LinkedList se implementa como una lista doblemente enlazada. Se puede
 * recorrer la lista con el ListIterator hacia adelante o hacia atras, pero
 * para obtener un elemento del medio el tiempo es proporcional al tamaño
 * de la lista. Su rendimiento en eliminar es mejor que el ArrayList,
 * pero es malo con los métodos get y set.
 *
 *   add es O(1)
 *   remove es O(n)
 *   set es O(n)
 *   get es O(n)
 *
 *   Con el iterator:
 *    remove es O(1)
 *
 */
public class EjemploLinkedList {
      @SuppressWarnings("unchecked")
      public static void main(String[] args) {
            LinkedList<Character> l = new LinkedList<Character>();

            /* ¿La lista esta vacía? */
            System.out.println(l.isEmpty());            // true

            l.add('a');                                 // ['a']
            l.add('b');                                 // ['a', 'b']
            l.add('c');                                 // ['a', 'b', 'c']
            System.out.println(l.size());               // 3
            System.out.println(l.get(1));               // b

            System.out.println(l);                      // [a, b, c]
            System.out.println(l.contains('b'));        // true
            System.out.println(l.indexOf('b'));         // 1
            System.out.println(l.lastIndexOf('b'));     // 1
            l.set(2, 'z');                              // ['a', 'b', 'z']
            l.remove(1);                                // ['a', 'z']

            /* Con iterador */
            ListIterator<Character> it = l.listIterator();
            it.next();                                  // Avanza
            it.add('y');                                // ['a', 'z', 'y']
            it.previous();                              // Retrocede
            it.add('x');                                // ['a', 'x', 'z', 'y']
            System.out.println(l);                      // [a, x, y, z]
            System.out.println(it.nextIndex());         // 2

            // Saca una copia
            LinkedList<Character> copia = (LinkedList<Character>) l.clone();

            System.out.println(copia);                  // [a, x, y, z]
            l.clear();                                  // Elimina todos los elementos.
            System.out.println(l.isEmpty());            // true
      }
}
```

LA CLASE COLLECTIONS

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

/**
 * La Clase Collections
 *
 */
public class EjemploColecciones {
	public static void main(String[] args) {
		Vector<Integer> v = new Vector<Integer>();

		/* ¿El arreglo esta vacío? */
		System.out.println(v.isEmpty());		// true

		v.add(100);					// [100]
		v.add(10);					// [100, 10]
		v.add(50);					// [100, 10, 50]

		Collections.sort(v);				// Ordena ascendentemente
		System.out.println(v);				// [10, 50, 100]

		int indice = Collections.binarySearch(v, 50); // Búsqueda binaria
		System.out.println(indice);			// 1

		Collections.reverse(v);				// Invierte
		System.out.println(v);				// [100, 50, 10]

		Collections.shuffle(v);				// Reordena aleatoriamente
		System.out.println(v);				// [n1, n2, n3]

		System.out.println(Collections.min(v));		// 10

		System.out.println(Collections.max(v));		// 100
	}
}
```

FORMAS DE RECORRIDO DE UN ARREGLO O LISTA

```
import java.util.ArrayList;
import java.util.Iterator;
/**
 * Recorrido de un ArrayList.
 *
 */
public class RecorridoArrayList {
	public static void main(String[] args) {
		ArrayList<String> a = new ArrayList<String>();

		a.add("a");
		a.add("b");
		a.add("c");

		System.out.println("#1 ciclo for normal");
		for (int i = 0; i < a.size(); i++) {
			System.out.println(a.get(i));
		}

		System.out.println("#2 ciclo while");
		int j = 0;
		while (j < a.size()) {
			System.out.println(a.get(j));
			j++;
		}

		System.out.println("#3 ciclo for avanzado");
		for (String temp : a) {
			System.out.println(temp);
		}

		System.out.println("#4 iterador (mejor)");
		Iterator<String> iterator = a.iterator();
		while (iterator.hasNext()) {
			System.out.println(iterator.next());
		}
	}
}
```