Compare, sort, search, autobox

**Comparable interface**

* Comparable interface is used to define the *natural order* of the objects of the class that implements it.
* One method that returns negative integer, zero or positive (<, = , >)

package java.lang;

public interface Comparable<T> {

public int compareTo(T o);

}

* Some collection classes, like TreeSet and TreeMap, store their elements in a sorted

order. You can specify the sort order of the elements by making their class implement the Comparable interface.

* It’s important to note that the implementation of method compareTo() should be consistent with the implementation of method equals(). This rule is recommended, but not required.

**Arrays**





The element stored at position from-Index is sorted, but the element stored at position toIndex isn’t

intArray = **new int**[]{20, 14, 4, 10, 5, 3};  
Arrays.*sort*(intArray, 1, 5); *//20 4 5 10 14 3*

* Arrays start indexing from 0 and toIndex is not included

**Collections**

* Class Collections defines two sorting methods to sort objects of List:

static <T extends Comparable<? super T>> void sort(List<T> list)

static <T> void sort(List<T> list, Comparator<? super T> c)

* Once sorted, new elements are added to a list according to the specific algorithm used by the underlying data structure. After you sort elements of an ArrayList, the new elements are added to its end.

**Binary search**

static int binarySearch(byte[] a, byte key)

static int binarySearch(int[] a, int key)

static int binarySearch(Object[] a, Object key)

static <T> int binarySearch(T[] a, T key, Comparator<? super T> c)

* Method binarySearch() returns the index of the search key, if it is contained in the list; otherwise it returns (-(insertion point) - 1). The insertion point is defined as the point at which the key would be inserted into the list.
* If sorting and searching using different elements in array, you may encounter ClassCastexception.