# Esercizio sui comparatori generici

Prendere il mio esempio di `insertionSort` basato sulle classi anonime e l'interfaccia `IntComparator` e farlo funzionare in Java.

Sulla carta fare uno schema di come è strutturato e come evolve nel tempo lo stack e lo heap.

Infine usare il debugger per verificare la propria ipotesi sull'evoluzione di stack e heap.

## Uso di comparator generici

```
package it.uniud.poo.abstractions;

/**
 * Example with parametric choice of sort direction.
 *
 */
public class MySortAnonymousClasses {

    /**
     * Labels to specify order direction
     */
    public  enum SortDirection {
        INCREASING,
        DECREASING
    }

    /**
     * Interface to be used as a parameter to
     * implement a particular kind of comparison
     * between int.
     */
    interface IntComparator {
        /**
         * @param x
         * @param y
         * @return true or false depending on what meaning we want to
         * give to compare. Eg. compare(x,y) can imply x<y, or x=2*y, or ...
         * In the context of sorting only x<y or x>y are useful choices.
         */
        boolean compare(int x, int y);
    }
```

```
/**
 * Run a simple example with two sorting procedures: up and down.
 */
public static void main (String a[]){
int[] arr1 = {10,34,2,56,7,67,88,42};

doParametricSort(arr1, SortDirection.INCREASING);
System.out.format("Increasing: ");
for(int i:arr1){
    System.out.print(i);
    System.out.print(", ");
}
doParametricSort(arr1, SortDirection.DECREASING);
System.out.format("\nDecreasing: ");
for(int i:arr1){
    System.out.print(i);
    System.out.print(", ");
    }
}
/**
 * sort the array a
 * MODIFY the array a so that values are ordered
 * @param a: an array of integers to be sorted
 * @param dir: the direction of the sort: INCREASING/DECREASING
 */
/**
 * @param a
 */
private static void doParametricSort(int[] a, SortDirection dir) {
    IntComparator ic = null; // the actual comparator that we will be using
    switch (dir) {
    case INCREASING:
        ic = new IntComparator() {
            @Override
            public boolean compare(int x, int y) {
                return (x < y);
            }
        };
        break;

    case DECREASING:
        ic = new IntComparator() {
                @Override
                public boolean compare(int x, int y) {
```

```
                    return (x > y);
                }
            };
            break;
    }

    doInsertionSort(a, ic);
}
/**
 * Sort the array a using the direction implied by the
 * generic comparator. MODIFY array a.
 * @param a
 * @param ic: a generic comparator for integers.
 */
private static void doInsertionSort(int[] a, IntComparator ic) {
    for (int i = 1; i < a.length; i++) {
        for (int j = i ; j > 0 ; j--){
            //if (a[j] < a[j-1]){
            if (ic.compare(a[j],a[j-1])){
                swap(a, j);
            }
        }
    }
}
/**
 * swap a[j] with a[j-1]; MODIFY the array a
 * @param a an array of integers, REQUIRED to have 2 or more elements
 * @param j an index of the array, REQUIRED to be a valid index and > 0.
 */
private static void swap(int[] a, int j) {
    int temp;
    temp = a[j];
    a[j] = a[j-1];
    a[j-1] = temp;
}
}
```