```java
import java.lang.reflect.Array;
import java.util.Comparator;

public class BinaryHeap<T extends Comparable<? super T>> {

    private static final int INITIAL_CAPACITY = 5;
    private T[] heap;
    private int heapSize;
    private Comparator<T> comp;
    private Class<T> type;

    @SuppressWarnings("unchecked")
    public BinaryHeap(Class<T> type) {
        this.type = type;
        // This is a workaround due to a limitation Java has with
        // constructing generic arrays.
        this.heap = (T[]) Array.newInstance(this.type, INITIAL_CAPACITY);
        this.heapSize = 0;
        this.comp = new NaturalComparator();
    }

    // Put methods for min-heap PQ implementation here.
    // Anywhere that you use x.compareTo(y) method to compare data of type T:
    //     instead, use comp.compare(x,y).
    // Then it's generalized so that we can do natural comparison (for min-heap)
    //     or reverse comparison (for max-heap)

    void sort(T[] array) {
        this.comp = new ReverseComparator(); // so we'll make/use a max heap rather than min heap
        T[] oldHeap = this.heap;
        int oldHeapSize = this.heapSize;
        this.heap = array;  // so the heap operations will work on the given array
        this.heapSize = array.length - 1; // assume array is full of data. Subtract 1 because
                                          // 0th entry is not part of the heap.

        // one step of selection sort to get smallest entry in the 0th spot

        // heapsort on this.heap, ignoring 0th entry
        // (A) buildHeap
        // (B) deleteMax N times, putting results at end from right to left

        this.heap = oldHeap; // return the heap field back to what it was
        this.heapSize = oldHeapSize;
        this.comp = new NaturalComparator();  // return back to min heap mode
    }

    class NaturalComparator implements Comparator<T> {
        @Override
        public int compare(T o1, T o2) {
        return o1.compareTo(o2);
        }
    }

    class ReverseComparator implements Comparator<T> {
        @Override
        public int compare(T o1, T o2) {
        return o2.compareTo(o1);
        }
    }
}
```