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

import components.queue.Queue;
import components.queue.Queue1L;
import components.simplewriter.SimpleWriter;
import components.simplewriter.SimpleWriter1L;

/**
 * Simple HelloWorld program (clear of Checkstyle and FindBugs warnings).
 *
 * @author Sam Espanioly
 */
public final class practice {

    /**
     * Default constructor--private to prevent instantiation.
     */
    private practice() {
        // no code needed here
    }

    /**
     * Reports the smallest integer in the given {@code Queue<Integer>}.
     *
     * @param q
     *            the queue of integer
     * @return the smallest integer in the given queue
     * @requires q /= empty_string
     * @ensures <pre>
     * min is in entries(q) and
     * for all x: integer
     *     where (x is in entries(q))
     *   (min <= x)
     * </pre>
     */
    private static int min(Queue<Integer> q) {
        int len = q.length();
        int minn = Integer.MAX_VALUE;
        Queue<Integer>[] x;
        // i need to practice using the for each loop but for now i use the regular for loop
        for (int i = 0; i < len; i++) {
            int temp = q.dequeue();
            if (temp < minn) {
                minn = temp;
            }
            q.enqueue(temp);

        }
        return minn;
    }

    //compareTo
    private static class IntegerLT implements Comparator<Integer> {
        @Override
        public int compare(Integer o1, Integer o2) {
            if (o1 < o2) {
                return -1;
            } else if (o1 > o2) {
                return 1;
            } else {
                return 0;
            }
        }
    }

    /**
     * Reports an array of two {@code int}s with the smallest and the largest
     * integer in the given {@code Queue<Integer>}.
     *
     * @param q
     *            the queue of integer
     * @return an array of two {@code int}s with the smallest and the largest
     *         integer in the given queue
     * @requires q /= empty_string
     * @ensures <pre>
     * { minAndMax[0], minAndMax[1] } is subset of entries(q) and
     * for all x: integer
     *     where (x in in entries(q))
     *   (minAndMax[0] <= x <= minAndMax[1])
     * </pre>
     */
    //using the ark algorithm
    private static int[] minAndMax(Queue<Integer> q) {
        // mnm = miminum n maximum
        int[] mnm = new int[2];
        int len = q.length();
        int min = Integer.MAX_VALUE;
        int max = Integer.MIN_VALUE;
        mnm[0] = min;
        mnm[1] = max;
        for (int i = 0; i < len; i++) {
            //minimum
            int temp1 = q.dequeue();
            //maximum
            int temp2 = q.dequeue();
            int com = Integer.compare(temp1, temp2);
            //switching temp number
            int temp3;
            //switching in case temp2 is smaller than temp1
            if (com > 0) {
                temp3 = temp1;
                temp1 = temp2;
                temp2 = temp3;
            }
            if (temp1 < mnm[0]) {
                mnm[0] = temp1;
            }
            if (temp2 > mnm[1]) {
                mnm[1] = temp2;
            }
            q.enqueue(temp1);
            q.enqueue(temp2);
        }
        return mnm;
    }

    /**
     * Main method.
     *
     * @param args
     *      the command line arguments; unused here
     */
    public static void main(String[] args) {
        SimpleWriter out = new SimpleWriter1L();
        Queue<Integer> x = new Queue1L<>();
        for (Integer i = 2; i < 20; i++) {
                x.enqueue(i);
            }
        //ensuring that the code will work
        x.enqueue(-20);
        x.enqueue(100);
        x.enqueue(-1);
        out.println(min(x));
        out.println(Arrays.toString(minAndMax(x)));
        //answers for the rest of the questions
        /*
         * Why do you need the requires clause? To make sure that the input is
         * possible and in the range of acceptable inputs and legal inputs Why
         * is the first line of the ensures clause important (min is in
         * entries(q))? Explain what the implementation could do if this line
         * was not included in the postcondition. I did not fully understand
         * that but min resembles all the entries of q but returns one of them
         */
        out.close();
    }

}
```