```java
/**
 * This class implements a set of integers.  It is limited by the size
 * sz chosen at compile time.  This is expected to be a good example
 * of software engineering in the tiny.  Possibly contains bugs.  We
 * naively assume all objects are non-null.  It is also intended as an
 * example for use with learning JUnit/TestNG, JML and FindBugs.

 * @file SmallSet.java
 * @author pmateti@wright.edu
 * @date Oct 20, 2013

 * <p> For further details, see
 * http://cecs.wright.edu/~pmateti/Courses/7140/Lectures
 * /Design/smallSet-DesignDoc.html; long names have been shortened.

 * <p>Apparantly javadoc has no tags for pre- and post-conditions; so
 * I am placing assert()s at the top of the methods for pre- and at
 * the bottom for post.  As is, we are not using JML syntax.  The old
 * objects, setOf() and classInv() are needed only because of asserts.
 * The obj1 == obj2 used below is intended to be deep equality.
 */

import java.io.*;

public class SmallSet {

    private int[] ear;          // element array that stores the set elements
    private int ux = 0;         // ear[0..ux-1] occupied; ear[ux] vacant
    private int sz = 1000;      // new-created size of this array

    /**
     * @param s any SmallSet
     * Class invariant, expressed as a boolean function.
     * @return truthhood of the boolen exp of the invariant
     */
    public boolean classInv(SmallSet s) {
        return
            0 <= s.ux && s.ux < s.sz
            && setOf(s) == setOf(s.ear, 0, ux-1)
            ;
    }

    /**
     * constructs an emptyset
     */

    public SmallSet() {
        ear = new int[sz];
    }

    public SmallSet(int [] a, int m, int n) {
        assert m < n;
        ear = new int[sz];
        for (int i = m; i < n; i++) {
            insert(a[i]);
        }
    }

    private SmallSet setOf(SmallSet s) {
        return s.compact();
    }

    private SmallSet setOf(int [] a, int m, int n) {
        return setOf(new SmallSet(a, m, n));
    }

    /**
```

```
     * @param e element to search for
     * @return any i such that ear[i] == e
     */
    private int indexOf(int e) {
        assert classInv(this);
        int i = 0;
        ear[ux] = e;
        while (ear[i] != e)
            i++;
        assert 0 <= i && i <= ux && e == ear[i] ;
        return i;
    }

    public boolean isin(int e) {
        int x = indexOf(e);
        assert x >= ux || (x == ear[x] && 0 <= x && x < ux);
        return (x < ux);
    }

    /**
     * While keeping the abstraction intact, compactify the ear[] so
     * that ux can be the lowest possible.  Rewrite it without using
     * newset.
     * @return the new SmallSet equal to setOf(this)
     */

    public SmallSet compact() {
        SmallSet newset = new SmallSet();
        while (ux > 0) {
            int e = ear[ux-1];
            delete(e);
            if (! newset.isin(e))
                newset.insert(e);
        }
        assert setOf(this) == setOf(newset) && newset.ux <= this.ux;
        return newset;
    }

    public SmallSet insert(int e) {
        if (ux < sz - 1) {
            if (ux > 0)
                ear[ux] = ear[0];
            ear[0] = e;
            ux++;
        } else {
            // see design obligations discussion at the URL given above.
        }
        assert ux == sz - 1 || isin(e);
        return this;
    }

    /**
     * Delete all occurrences of e in ear[a to b-1], and compact ear[].
     * A casual reader comments: This is tricky!  Do write a loop invariant.
     * @return the count of deletions.
     */
    private int delete(int a, int b, int e) {
        assert 0 <= a && a < b;
        int nd = 0;             // deleted e this many times
        for (int i = a, j = a; i < b; i++) {
            if (ear[i] == e)
                nd ++;
            else {
                if (j < i)
                    ear[j] = ear[i];
                j++;
            }
```

```java
        }
        assert 0 <= nd && nd < b - a && ! setOf(ear, a, b).isin(e);
        return nd;
    }

    /**
     * @param e All occurrences of e in ear[] must be deleted.
     * @return this
     */

    public SmallSet delete(int e) {
        assert ux < sz;
        ux -= delete(0, ux, e);
        assert ux < sz && !isin(e);
        return this;
    }

    /**
     * @return size of the set; Side effect: this set gets compacted;
     */
    public int nitems() {
        SmallSet s = compact();
        ux = s.ux;
        ear = s.ear;
        assert ux == setOf(s).nitems();
        return ux;
    }

    public SmallSet union(SmallSet tba) {
        SmallSet old = this;
        for (int i = 0; i < tba.ux; i++)
            insert(tba.ear[i]);
        assert setOf(this) == setOf(old).union(setOf(tba));
        return this;
    }

    public SmallSet diff(SmallSet tbs) {
        SmallSet old = this;
        SmallSet newset = new SmallSet();
        for (int i = 0; i < tbs.ux; i++)
            if (! this.isin(tbs.ear[i])) newset.insert(tbs.ear[i]);
        for (int i = 0; i < this.ux; i++) {
            if (! tbs.isin(this.ear[i])) newset.insert(this.ear[i]);
        }
        ear = newset.ear;
        ux = newset.ux;
        assert setOf(this) == setOf(old).diff(setOf(tbs));
        return this;
    }

    public String toString() {
        String s = "el: ";
        for (int i = 0; i < ux; i++) {
            s += ", " + ear[i];
        }
        s += ", ux=" + ux;
        return s;
    }

    public static void main(String[] args) {
        SmallSet s = new SmallSet();
        SmallSet t = new SmallSet();
        int [] a = {1,2,3,4,5,6};
        for (int j = 0; j < 3; j++)
            for (int i = 0; i < a.length; i++) { // or use setOf()
                s.insert(a[i]);
                t.insert(a[i] - 1);
            }
        // some simple tests
        System.out.println("set s " + s + "; nitems=" + s.nitems());
        s.delete(1);
        s.delete(3);
        System.out.println("set s " + s + "; nitems=" + s.nitems());
        s.union(t);
        System.out.println("set s " + s + "; nitems=" + s.nitems());
        t.insert(7);
        t.diff(s);
        System.out.println("set t " + t + "; nitems=" + t.nitems());
    }
}

// -eof-
```