```java
package caltool.schedule;

import caltool.caldb.*;
import mvp.*;
import java.util.*;

/****
 *
 * Class ScheduleTest is the companion testing class for class <a href=
 * Schedule.html> Schedule </a>.  It implements the following module test plan:
 *                                                                          <pre>
 *     Phase 1: Unit test the constructor.
 *
 *     Phase 2: Unit test the simple access method getCategories.
 *
 *     Phase 3: Unit test the constructive methods scheduleAppointment,
 *              scheduleTask, and scheduleEvent.
 *
 *     Phase 4: Unit test the constructive methods scheduleMeeting and
 *              confirmMeeting.
 *
 *     Phase 5: Unit test the changeItem and deleteItem methods.
 *
 *     Phase 6: Repeat phases 1 through 5.
 *
 *     Phase 7: Stress test by scheduling and deleting 100000 items.
 *                                                                         </pre>
 */
public class ScheduleTest extends Schedule {

    /*-*
     * Public methods
     */

    /**
     * Construct by calling the parent constructor with a null View and the
     * given stubbed CalendarDB.
     */
    public ScheduleTest(CalendarDB calDB) {
        super(null, calDB);
    }

    /**
     * Run all the phases of this.
     */
    public void run() {
        phase1();
        phase2();
        phase3();
        phase4();
        phase5();
        phase6();
        phase7();
    }


    /*-*
     * PhaseX methods are for each module testing phase.
     */

    /**
     * Execute test phase 1 by calling testSchedule.
     */
    protected void phase1() {
        dumpPhaseHeader(1);
        testSchedule();
        dumpPhaseEndSpacing();
    }

    /**
     * Execute test phase 2 by calling testGetCategories.
     */
    protected void phase2() {
        dumpPhaseHeader(2);
        testGetCategories();
        dumpPhaseEndSpacing();
    }

    /**
     * Execute test phase 3 by calling testScheduleAppointment,
     * testScheduleTask, and testScheduleEvent on the schedule built in phase
     * 1.  These three unit tests in turn exercise the testAlreadyScheduled
     * and testValidateInputs methods.
     */
    protected void phase3() {
        dumpPhaseHeader(3);
        testScheduleAppointment();
        testScheduleTask();
        testScheduleEvent();
        dumpPhaseEndSpacing();
    }

    /**
     * Execute test phase 4 by ...
     */
    protected void phase4() {
        dumpPhaseHeader(4);
        /* ... */
        dumpPhaseEndSpacing();
    }

    /**
     * Execute test phase 5 by ...
     */
    protected void phase5() {
        dumpPhaseHeader(5);
        /* ... */
        dumpPhaseEndSpacing();
    }

    /**
     * Execute test phase 6 by ...
```

```
     */
    protected void phase6() {
        dumpPhaseHeader(6);
        /* ... */
        dumpPhaseEndSpacing();
    }


    /**
     * Execute test phase 7 by ...
     */
    protected void phase7() {
        dumpPhaseHeader(7);
        /* ... */
        dumpPhaseEndSpacing();
    }

    /*-*
     * Individual unit testing methods for member methods
     */

    /**
     * Unit test the constructor by building one Schedule object.  No further
     * constructor testing is necessary since only one Schedule object is ever
     * constructed in the CalendarTool system.
     */
    protected void testSchedule() {
        dumpUnitTestHeader("Constructor");
        schedule = new Schedule(null, calDB);
        dumpUnitTestEndSpacing();
    }

    /**
     * Unit test getCategories by calling getCategories on a schedule with a
     * null and non-null categories field.  The Categories model is tested
     * fully in its own class test.
     *                                                                   <pre>
     *  Test
     *  Case    Input                   Output          Remarks
     * ====================================================================
     *   1      schedule.categories     null            Null case
     *            = null
     *
     *   2      schedule.categories     same non-null   Non-null case
     *            = non-null value      value
     *                                                                  </pre>
     */
    protected void testGetCategories() {

        /*
         * Dump a unit test header.
         */
        dumpUnitTestHeader("getCategories");

        /*
         * Do case 1 and dump the results.
         */
        schedule.categories = null;
        dump("Case 1:");
        dump("Should be null: " + schedule.getCategories());

        /*
         * Do case 2 and dump the results.
         */
        schedule.categories = new Categories();
        dump("Case 2:");
        dump("Should be non-null: " + schedule.getCategories());

        dumpUnitTestEndSpacing();
    }

    /**
     * Unit test scheduleAppointment by ...
     */
    protected void testScheduleAppointment() {
        /* ... */
    }

    /**
     * Unit test scheduleAppointment by ...
     */
    protected void testScheduleTask() {
        /* ... */
    }

    /**
     * Unit test scheduleEvent by supplying inputs that test the value range of
     * each Event data field and exercise each precondition clause.  The cases
     * are as follows:
     *                                                                   <pre>
     *  Test
     *  Case    Input                   Output          Remarks
     * ====================================================================
     *   1      {"Event 0",             Input event     Lower end of possible
     *           {"Sunday",1,           added to items  event data range
     *            "January",1},         no junk, no
     *            null,                 confusion
     *            null,
     *            null}
     *
     *   2      {10000-char string,     Input event     Upper end of possible
     *           {"Sunday",1,           added to items  event data range
     *            "January",1},         no junk,
     *           {"Saturday",31,        no confusion
     *            "December", 9999},
     *           10000-char string
     *             reversed,
     *           10000-char string}
     *
     *   3      events with start and   Input event     Events with all combos
     *  thru    end date ranging thru   added to items  of days, dates, and
     *  2564    each legal day, each    no junk, no     months
```

```
     *         legal date, and each    confusion
     *         legal month, with
     *         arbitrary year,
     *         category, and security
     *
     *  2565   32 cases to exercise    Schedule        Events that violate
     *  thru   possible boolean        Event           precond logic.
     *  2597   values for 5 precond    Precond
     *         clauses                 Exception
     *                                  thrown
     *                                                                  </pre>
     */
    protected void testScheduleEvent() {

        /*
         * Dump a unit test header.
         */
        dumpUnitTestHeader("scheduleEvent");

        /*
         * Do the cases.
         */
        try {
            testScheduleEventCase1();
            testScheduleEventCase2();
            testScheduleEventCases3Thru2564();
            testScheduleEventCases2565Thru2597();
        }
        catch (ScheduleEventPrecondViolation e) {
            /* This will never happen */
        }

        /*
         * Dump the test case results.
         */
        dump("");
        dumpUnitTestEndSpacing();

    }

    /**
     * Run testScheduleEvent case 1.
     */
    protected void testScheduleEventCase1()
            throws ScheduleEventPrecondViolation {

        dumpUnitTestCaseHeader("scheduleEvent", 1);
        schedule.scheduleEvent(new Event(
            "Event 0",                          // title
            new Date(new DayName("Sunday"),     // start date
                1,
                new MonthName("January"),
                1),
            new Date(new DayName("Tuesday"),    // start date
                2,
                new MonthName("January"),
                1),
            new Category("Category 1"),         // category
            new SimpleSecurity("public")                // security
        ));
    }

    /**
     * Run testScheduleEvent case 2.
     */
    protected void testScheduleEventCase2()
            throws ScheduleEventPrecondViolation {

        int i;
        String longString = "";
        String longStringReversed = "";
        for (i = 0; i < 10000; i++) {
            longString += (char) i % 256;
            longStringReversed = (char) i + longStringReversed;
        }

        dumpUnitTestCaseHeader("scheduleEvent", 2);

        schedule.scheduleEvent(new Event(
            longString,                         // title
            new Date(new DayName("Sunday"),     // start date
                1,
                new MonthName("January"),
                1),
            new Date(new DayName("Tuesday"),    // start date
                2,
                new MonthName("January"),
                1),
            new Category(longStringReversed),   // category
            new SimpleSecurity("private")       // security
        ));
    }

    /**
     * Run testScheduleEvent cases 3 through 2564 by looping through various
     * combinations of days, weeks, and months.
     */
    protected void testScheduleEventCases3Thru2564()
            throws ScheduleEventPrecondViolation {

        int day, number, month, year, i, n;
        Date date;

        for (i = 3, day = 0; day < 7; day++) {
            for (number = 1; number <= 31; number++) {
                for (month = 0; month < 12; month++) {

                    date = new Date(
                        new DayName(day),
                        number,
                        new MonthName(month),
                        year = number * 100);
```

```java

                    if (date.isValid()) {
                        dumpUnitTestCaseHeader("scheduleEvent", i++);

                        schedule.scheduleEvent(new Event(
                            "Event " +                              // title
                                String.valueOf(n = year + month + day),
                            date,                                   // start
                            date,                                   // end
                            new Category(String.valueOf(n*10)),     // category
                            ((number % 2) == 0) ?                   // security
                                new SimpleSecurity("public") :
                                new SimpleSecurity("private")
                        ));
                    }
                }
            }
        }

    }

    /**
     * Run testScheduleEvent cases 2565 through 2597 to fully exercise the
     * precond logic.
     */
    protected void testScheduleEventCases2565Thru2597() {
        /* ... */
    }


    /*-*
     * Testing utility methods.
     */

    /**
     * Output a header message to stdout identifying the test phase.
     */
    protected void dumpPhaseHeader(int phasenum) {
        System.out.print("**** Schedule Testing Phase ");
        System.out.print(phasenum);
        System.out.println(" ****");
    }

    /**
     * Output a header message to stdout for the unit test of the given
     * testname.
     */
    protected void dumpUnitTestHeader(String testname) {
        System.out.print("** Unit Test " + testname + " **\n");
    }

    /**
     * Output a header message to stdout for a unit test case.
     */
    protected void dumpUnitTestCaseHeader(String testname, int caseNumber) {
        System.out.print("* Unit Test Case " +
            testname + String.valueOf(caseNumber)  + " **\n");
    }


    /**
     * Dump three blank lines following a test phase.
     */
    protected void dumpPhaseEndSpacing() {
        System.out.print("\n\n\n");
    }

    /**
     * Dump a couple blank lines following a unit test.
     */
    protected void dumpUnitTestEndSpacing() {
        System.out.print("\n\n");
    }

    /**
     * Dump the data values of schedule to stdout.  For test validation
     * purposes, include a print of the number of elements in the dumped items.
     * Precede the dump with the given message, if the message is non-null.
     */
    protected void dump(String message) {
        int l;                  // Temp

        if (message != null) {
            System.out.print("* " + message + " *" + "\n");
        }

        System.out.print("Schedule contains\n" +
            "Categories: " + schedule.categories + "\n" +
            (l = schedule.calDB.getCurrentCalendar().numItems()) + " " +
            (l == 1 ? " item" : " items") + "\n" +
            schedule.toString() + "\n"
        );
    }

    /**
     * Schedule data object to support the tests.
     */
    protected Schedule schedule;

}
```