```
1  package caltool.schedule;
2
3  import caltool.caldb.*;
4  import mvp.*;
5  import java.util.*;
6
7  /****
8   *
9   * Class ScheduleTest is the companion testing class for class <a href=
10  * Schedule.html> Schedule </a>.  It implements the following module test plan:
11  *                                                                      <pre>
12  *     Phase 1: Unit test the constructor.
13  *
14  *     Phase 2: Unit test the simple access method getCategories.
15  *
16  *     Phase 3: Unit test the constructive methods scheduleAppointment,
17  *              scheduleTask, and scheduleEvent.
18  *
19  *     Phase 4: Unit test the constructive methods scheduleMeeting and
20  *              confirmMeeting.
21  *
22  *     Phase 5: Unit test the changeItem and deleteItem methods.
23  *
24  *     Phase 6: Repeat phases 1 through 5.
25  *
26  *     Phase 7: Stress test by scheduling and deleting 100000 items.
27  *                                                                    </pre>
28  */
29 public class ScheduleTest extends Schedule {
30
31     /*-*
32      * Public methods
33      */
34
35     /**
36      * Construct by calling the parent constructor with a null View and the
37      * given stubbed CalendarDB.
38      */
39     public ScheduleTest(CalendarDB calDB) {
40         super(null, calDB);
41     }
42
43     /**
44      * Run all the phases of this.
45      */
46     public void run() {
47         phase1();
48         phase2();
49         phase3();
50         phase4();
51         phase5();
52         phase6();
53         phase7();
54     }
55
56
57     /*-*
58      * PhaseX methods are for each module testing phase.
59      */
60
61     /**
62      * Execute test phase 1 by calling testSchedule.
63      */
64     protected void phase1() {
65         dumpPhaseHeader(1);
66         testSchedule();
67         dumpPhaseEndSpacing();
68     }
69
70     /**
71      * Execute test phase 2 by calling testGetCategories.
72      */
73     protected void phase2() {
74         dumpPhaseHeader(2);
75         testGetCategories();
76         dumpPhaseEndSpacing();
77     }
78
79     /**
80      * Execute test phase 3 by calling testScheduleAppointment,
81      * testScheduleTask, and testScheduleEvent on the schedule built in phase
82      * 1.  These three unit tests in turn exercise the testAlreadyScheduled
83      * and testValidateInputs methods.
84      */
85     protected void phase3() {
86         dumpPhaseHeader(3);
87         testScheduleAppointment();
88         testScheduleTask();
89         testScheduleEvent();
90         dumpPhaseEndSpacing();
91     }
92
93     /**
94      * Execute test phase 4 by ...
95      */
96     protected void phase4() {
97         dumpPhaseHeader(4);
98         /* ... */
99         dumpPhaseEndSpacing();
100    }
101
102    /**
103     * Execute test phase 5 by ...
104     */
105    protected void phase5() {
106        dumpPhaseHeader(5);
107        /* ... */
108        dumpPhaseEndSpacing();
109    }
110
111    /**
112     * Execute test phase 6 by ...
```

```
113      */
114     protected void phase6() {
115         dumpPhaseHeader(6);
116         /* ... */
117         dumpPhaseEndSpacing();
118     }
119
120
121     /**
122      * Execute test phase 7 by ...
123      */
124     protected void phase7() {
125         dumpPhaseHeader(7);
126         /* ... */
127         dumpPhaseEndSpacing();
128     }
129
130     /*-*
131      * Individual unit testing methods for member methods
132      */
133
134     /**
135      * Unit test the constructor by building one Schedule object.  No further
136      * constructor testing is necessary since only one Schedule object is every
137      * constructed in the CalendarTool system.
138      */
139     protected void testSchedule() {
140         dumpUnitTestHeader("Constructor");
141         schedule = new Schedule(null, calDB);
142         dumpUnitTestEndSpacing();
143     }
144
145     /**
146      * Unit test getCategories by calling getCategories on a schedule with a
147      * null and non-null categories field.  The Categories model is tested
148      * fully in its own class test.
149      *                                                                    <pre>
150      *  Test
151      *  Case    Input                   Output          Remarks
152      * ========================================================================
153      *   1      schedule.categories     null            Null case
154      *            = null
155      *
156      *   2      schedule.categories     same non-null   Non-null case
157      *            = non-null value      value
158      *                                                                   </pre>
159      */
160     protected void testGetCategories() {
161
162         /*
163          * Dump a unit test header.
164          */
165         dumpUnitTestHeader("getCategories");
166
167         /*
168          * Do case 1 and dump the results.
169          */
170         schedule.categories = null;
171         dump("Case 1:");
172
173         /*
174          * Do case 2 and dump the results.
175          */
176         schedule.categories = new Categories();
177         dump("Case 2:");
178
179     }
180
181     /**
182      * Unit test scheduleAppointment by ...
183      */
184     protected void testScheduleAppointment() {
185         /* ... */
186     }
187
188     /**
189      * Unit test scheduleAppointment by ...
190      */
191     protected void testScheduleTask() {
192         /* ... */
193     }
194
195     /**
196      * Unit test scheduleEvent by supplying inputs that test the value range of
197      * each Event data field and exercise each precondition clause.  The cases
198      * are as follows:
199      *                                                                    <pre>
200      *  Test
201      *  Case    Input                   Output          Remarks
202      * ========================================================================
203      *   1      {"Event 0",             Input event     Lower end of possible
204      *           {"Sunday",1,           added to items  event data range
205      *            "January",1},         no junk, no
206      *           null,                   confusion
207      *           null,
208      *           null}
209      *
210      *   2      {10000-char string,     Input event     Upper end of possible
211      *           {"Sunday",1,           added to items  event data range
212      *            "January",1},         no junk,
213      *           {"Saturday",31,         no confusion
214      *            "December", 9999},
215      *           10000-char string
216      *             reversed,
217      *           10000-char string}
218      *
219      *   3      events with start and   Input event     Events with all combos
220      *  thru    end date ranging thru   added to items  of days, dates, and
221      *  2564    each legal day, each    no junk, no     months
222      *          legal date, and each    confusion
223      *          legal month, with
224      *          arbitrary year,
```

```
 *          category, and location
 *
 *  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
        "Location 1"                        // location
    ));
}

/**
 * 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
        longString                          // location
    ));
}

/**
 * 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);

                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
                            "Location " + String.valueOf(n*100)         // location
                        ));
                    }
                }
            }
        }

    }

    /**
     * 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()) + "\n" +
            (l == 1 ? " item" : " items") + "\n" +
            schedule.toString() + "\n"
        );
    }

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

}
```