A Small JUnit Example

The faulty code in Fig. 1 is from the clock driver for the Freescale MC 13783 processor used by the Microsoft Zune 30 and the Toshiba Gigabeat S media players.[1] On the last day of a leap year, the code loops forever. On December 31, 2008, owners of the Zune media player therefore awoke to find that it froze on startup. The root cause of the failure was isolated by "itsnotabigtruck" [2].

```
1)  year = ORIGINYEAR; /* = 1980 */
2)  while (days > 365)
3)  {
4)      if (IsLeapYear(year))
5)      {
6)          if (days > 366)
7)          {
8)              days -= 366;
9)              year += 1;
10)         }
11)     }
12)     else
13)     {
14)         days -= 365;
15)         year += 1;
16)     }
17) }
```

Figure 1: Where is the fault?

A Testable Unit: Class `Date.java`

The code fragment in Fig. 1 cannot be tested by itself since it is not self contained: it needs a value for `ORIGINYEAR` and an implementation of function `IsLeapYear()`. Class `Date.java` is a self-contained version of the code in Fig. 1:

```
1) package src;
2) import java.io.*;
3)
```

```
 4) public class Date {
 5)     public static int ORIGINYEAR = 1980;
 6)
 7)     public boolean IsLeapYear(int year) {
 8)         if( year % 4 != 0 )
 9)             return false;
10)         if( year % 100 != 0 )
11)             return true;
12)         return year % 400 == 0;
13)     }
14)
15)     public int getYear(int days) {
16)
17)         int year = ORIGINYEAR;
18)         while (days >  365 )
19)         {
20)             if (IsLeapYear(year))
21)             {
22)                 if (days > 366)
23)                 {
24)                     days -= 366;
25)                     year += 1;
26)                 }
27)             }
28)             else
29)             {
30)                 days -= 365;
31)                 year += 1;
32)             }
33)         }
34)         return year;
35)     }
36) }
```

Some JUnit Tests

Convenient automated unit testing profoundly changes software development. A full suite of tests can be run automatically at any time to verify the code. Code and tests can be developed together; new tests can be added as development proceeds. In fact, automated tests enable test-first or test-driven development, where tests are written first and then code is written to pass the tests.

Convenience was the number one goal for *JUnit*, a framework for automated testing of Java programs. Kent Beck and Erich Gamma wanted to make it so convenient that "we have some glimmer of hope that developers will actually write tests."[2]

A JUnit test proceeds as follows:

set up the context for the code under test;
run test;
evaluate response;
tear down the context, if needed;

The annotation `@Test` marks the start of a test. The code to be tested is function `getYear()` in class `Date`. The name of the following test is `test365a`.

```
@Test
public void test365a() {
    Date date = new Date();           // set up context
    int year = date.getYear(365);     // run test
    assertEquals( year, 1980 );       // evaluate response
}
```

At the risk of being redundant, this test sets up the context by creating object `date` of class `Date`. It runs the test by calling `date.geatYear(365)`. The assertion `assertEquals()` checks that the computed value of `year` matches the expected value `1980` (`assertEquals()` is just one of the many assertions supported by JUnit).

This test is one of the four tests in file `DateTest.java` (see below). The four tests were chosen to illustrate how JUnit handles successful tests, failed tests, and infinite loops:

`test365a`	success: the computed value equals the expected value
`test365b`	test fails
`test366`	infinite loop times out
`test367`	another successful test

In the following code for file `DateTest.java`, note that the annotation for test `test366` has a timeout parameter:

```
25)     @Test(timeout=250)
```

The parameter `250` specifies that the test times out in 250 milliseconds. We could have added a timeout parameter to each of the tests.

```
 1) package test;
 2)
 3) import org.junit.*;
 4) import static org.junit.Assert.*;
 5)
 6) import java.io.*;
 7) import src.*;
 8)
 9) public class DateTest {
10)
11)     @Test
12)     public void test365a() {
13)         Date date = new Date();
14)         int year = date.getYear(365);
15)         assertEquals( year, 1980 );
16)     }
17)
18)     @Test
19)     public void test365b() {
20)         Date date = new Date();
21)         int year = date.getYear(365);
22)         assertEquals( year, 1981 );
23)     }
```

```
24)
25)     @Test(timeout=250)
26)     public void test366() {
27)         Date date = new Date();
28)         int year = date.getYear(366);
29)         assertEquals( year, 1980 );
30)     }
31)
32)     @Test
33)     public void test367() {
34)         Date date = new Date();
35)         int year = date.getYear(367);
36)         assertEquals( year, 1981 );
37)     }
38) }
```

Running Tests and Reporting Results

Where are we? Class `Date` contains the software to be tested. Class `DateTest` defines tests for method `getYear()` in `Date`.

It's time to run the tests and report on the results. Lines 13-15 of file `TestRunner.java` do just that:

```
 1) package test;
 2)
 3) import org.junit.runner.JUnitCore;
 4) import org.junit.runner.Result;
 5) import org.junit.runner.notification.Failure;
 6)
 7) import src.*;
 8)
 9) public class TestRunner {
10)     public static void main(String[] args) {
11)
12)         Result result = JUnitCore.runClasses(DateTest.class);
13)         for (Failure failure : result.getFailures()) {
14)             System.out.println(failure.toString());
15)         }
16)     }
17) }
```

Transcript of a Session

For this transcript, the file structure is as in Fig. 2. The code for JUnit happens to be in directory `$DIR`:

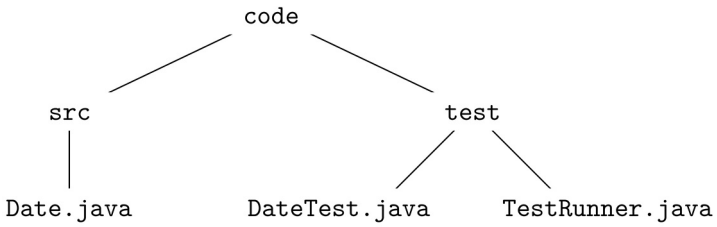

Figure 2: In this example, the source code to be tested is in directory `src`; the tests are in directory `test`.

```
Last login: Tue Oct  3 20:41:10 on ttys000
Kolea:~ Ravi$ export DIR=/Users/Ravi/Work/Committee/210/Exercises
```

We need to tell JUnit where to find files (the path for the directory `code` in Fig. 2 is in `$DIR/zune/code`):

```
Kolea:~ Ravi$ export JAVA_HOME=/Library/Java/Home
Kolea:~ Ravi$ export JUNIT_HOME=$DIR/JUnit
Kolea:~ Ravi$ export CLASSPATH=:$JUNIT_HOME/junit-4.12.jar
Kolea:~ Ravi$ export CLASSPATH=$CLASSPATH:/$JUNIT_HOME/hamcrest-core-1.3.jar
Kolea:~ Ravi$ export CLASSPATH=$CLASSPATH:/$DIR/zune/code
```

At this point, we're ready to run the tests:

```
Kolea:~ Ravi$ cd $DIR/zune/code
Kolea:code Ravi$ (cd src; javac *.java)
Kolea:code Ravi$ (cd test; javac *.java)
Kolea:code Ravi$ java test/TestRunner
```

The output shows only the failed tests:

```
test366(test.DateTest): test timed out after 250 milliseconds
test365b(test.DateTest): expected:<1980> but was:<1981>
```

Showing only failed tests is a good thing. There would be many more tests if we had designed tests with code coverage in mind. Messages about successful tests would clutter up the output.

Notes

[1]The Wikipedia article [3] has a section entitled "Freescale Driver Issue."

[2]The xUnit family began with Kent Beck's automated testing frameworks for Smalltalk. In 1997, Smalltalk usage was on the decline and Java usage was on the rise, so Kent Beck and Erich Gamma created JUnit for Java. They had three goals for JUnit: make it natural enough that developers would actually use it; enable tests that retain their value over time; and leverage existing tests in creating new ones [1].

References

1. Kent Beck and Erich Gamma. JUnit: a cook's tour. (circa 1998). `http://junit.sourceforge.net/doc/cookstour/cookstour.htm`.
2. itsnotabigtruck. Cause of Zune 30 leapyear problem ISOLATED! `http://www.zuneboards.com/forums/showthread.php?t=38143`.
3. Wikipedia. Zune 30. `https://en.wikipedia.org/wiki/Zune_30`.