Regression Testing

Goal

Regression tests validate that a program still produces "correct" output after the program has been updated. A common form of regression testing compares the **System.out** output of a run to a known correct output. Many of our Java programs benefit from such tests. **JUnit** is a standard for regression testing.

Problem

A problem with **JUnit** is that it is a test harness to create input to a method and to capture the output of a method call and compare it to the correct results. You need more when the output to be captured is via **System.out.**

Solution Regression Testing System.out

Here is how to create these tests. The instructions below are targeted for **NetBeans**, but can be used for other IDEs as well. You will need the following JAR file and if you are interested, its Java code:

- RegTest.jar
- NetBeans Project (optional, FYI)

In **NetBeans**, create a **JUnit** file:

- right click on the Java file to create a test→tools→create JUnit Tests; OK. This creates a **JUnit** test for the particular Java file you selected. But be careful -- If you have a test file already created, recreating it deletes the old file and installs a stub of a new test file.
- right click on Test Libraries→Add JAR/Folder→select "RegTest.jar". This adds "RegTest.jar" to the list of additional files you need to compile your test files.

Now, transform the test that was generated, which might look like this:

```
public void testMain() {
   System.out.println("main");

   String[] args = null;
   Main.main(args);

   // TODO review the generated test code and remove the default call to fail.
   fail("The test case is a prototype.");
}
```

to:

```
public void testMain() {
   RegTest.Utility.redirectStdOut("out.txt");  // redirects standard out to file "out.txt"

   String[] args = null;
   Main.main(args);

   RegTest.Utility.validate("out.txt", "correctOut.txt", false); // test passes if files are equal
}
```

where "**out.txt**" is the name of the file to which all **System.out** is to be written, and "**correctOut.txt**" is the name of the file that contains the correct output against which the "**out.txt**" is compared. "**false**" means don't sort the files before comparing -- sorting is useful if the order in which lines are output does not matter.

To execute a test program in **NetBeans**: Run→TestProject.

Note: this jar file requires the use of a JUnit jar file. Netbeans comes with the lastest JUnit library, so you need not download anything.

New: Advanced Regression Testing

Occasionally, you have files with generated values (identifiers, x-y coordinates) that may not match but have no relevance to equality. So to do a diff on these files, you first have to remove the irrelevant data. Here are two files, side by side. They are clearly not identical because the manufactured identifiers given in tuples are different. But these identifiers are not really essential to invalidate equality.

output.txt	correct.txt
dbase(prog1,[bcClass,bcMember]). table(bcClass,[cid,"name","superName"]). table(bcMember,[mid,cid,static,"type","sig"]). bcClass(d0,'p1package.ReflectionTest','Object'). bcMember(n0,c0,true,'p1package.ReflectionTest','ReflectionTest(int,int,int)'). bcMember(n1,c0,false,'double','d0'). bcMember(n2,c0,false,'double[]','d1'). bcMember(n3,c0,false,'double[][]','d2'). bcMember(n4,c0,false,'double[][][]','d3').	dbase(prog1,[bcClass,bcMember]). table(bcClass,[cid,"name","superName"]). table(bcMember,[mid,cid,static,"type","sig"]). bcClass(c0,'p1package.ReflectionTest','Object'). bcMember(m0,c0,true,'p1package.ReflectionTest','ReflectionTest(int,int,int bcMember(m1,c0,false,'double','d0'). bcMember(m2,c0,false,'double[]','d1'). bcMember(m3,c0,false,'double[][]','d2'). bcMember(m4,c0,false,'double[][][]','d3').

```
bcMember(n5,c0,true,'int','i1').
bcMember(n6,c0,true,'void','main(String[])').
bcMember(n7,c0,false,'String[][]','test(int[])').
bcMember(n8,c0,true,'void','main2(String[])').
```

```
bcMember(m5,c0,true,'int','i1').
bcMember(m6,c0,true,'void','main(String[])').
bcMember(m7,c0,false,'String[][]','test(int[])').
bcMember(m8,c0,true,'void','main2(String[])').
```

There is a facility in RegTest.Utility that allows you to specify an array of java regular expressions (Strings) that are eliminated from both the "correct" file and you "output" file before diffing takes place. The method is defined as:

```
public static void validate(String outputFile, String correctFile, boolean sortedTest, String[] eliminate)
```

To compare the above files (sans identifiers) is accomplished by the following call:

```
static String[] eliminate = {"\\(m.","\\(c.","\\(d.","\\(n.","};

@Test
public void testSomeMethod() {
   Utility.validate("output.txt", "correct.txt", false, eliminate);
}
```

The eliminate array removes a parenthesis, 2 characters, and comma with nothing from both files, and then compares the result (to say that they are equal).

Regression Testing Other Output Files

Sometimes programs produce file output, in addition to terminal output. You want to check the contents of these files too in regression testing. Use the same testing harness except with different calls. Simply use one or more calls to **RegTest.Utility.validate()** to compare outputted files with their correct counterparts. You don't need to use **RegTest.Utility.redirectStdOut()** if you are not interested in capturing standard out output. So the general format of a test is:

```
// this call is optional -- use only if you want to capture standard out
RegTest.Utility.redirectStdOut("stdout.txt"); (place output in file "stdout.txt")

{run program};

// now for every file that is produced, you can compare it to its correct output:
RegTest.Utility.validate("stdout.txt", "correctStdOut.txt", false); // false means don't sort before comparing files
RegTest.Utility.validate("A.txt", "correctA.txt", true);            // true means sort both files before comparing
RegTest.Utility.validate("B.txt", "correctB.txt", true);
```

Advice

Generally, you may not know the correct output of your program. What can do in such circumstances is to run the program, collect its output, and manually check to see if the output is correct. Once it is correct, you can simply rename the output file as "correct", and use it in your tests above. Remember: when you are performing multiple tests, NetBeans will simply count the number of tests that have succeeded and failed. So looking at any output files that may have been built along the way won't help you (as you won't know from which test the file was produced). By developing and debugging tests one at a time, and finding their correct output, you'll save yourself a lot of headaches.