JUnit Testing

DrJava has built-in mechanism that makes it easy to create a class to maintain a suite of test cases and to run *all* test cases whenever appropriate just by clicking a button. The mechanism, which is used in many interactive development environments, is called JUnit.

We now show how to construct test cases using a JUnit tester.

Class Panda

In DrJava, we have a class `Panda`, each instance of which maintains information about a panda. To keep things simple here, the class maintains only the panda's name and its father, if the father is known. In a real situation, much more information would be maintained. There is a constructor, two getter methods, and two setter methods. They must all be tested, and we do this using a JUnit testing class that saves all test cases.

Creating a JUnit testing class

To create a JUnit class, select menu **File** item **new JUnit test case…**. At the prompt, type a class name for this tester. We recommend the name of the class being tested followed by the word "Tester". Click button *OK*.

The new class has been created. Save it in the same folder as class `Panda`.

The import statement indicates that a class `TestCase` appears in package `junit.framework`, and class `PandaTester` extends `TestCase`. Several methods are inherited from class `TestCase`. We'll show the use of one of them later.

Class `PandaTester` comes with one procedure, `TestX`. As its comment says, we can replace the `X` by our own name, and we replace it by `ConstructorGetters`.

We can have as many test procedures as we wish. We add a second one for testing the setter methods.

Writing a test method

Now let's write the body of the procedure to test the constructor and getter methods. First, we need to create some Panda objects; let's use these two assignments:

```
Panda p1= new Panda("Shuaung", null);
Panda p2= new Panda("Lin", p1);
```

Note that we have created one without a father and one with a father in order to test these two different cases.

To test whether object `p1` has been correctly created, we need to check that each field of `p1` has the right value. Inherited procedure `assertEquals` can be used for this purpose. This procedure tests whether its two parameters are equal. The first parameter is the expected value, and the second is a value that is produced by the method being tested.

Here, we write two calls on `assertEquals`, testing whether the getter methods in object `p1` produce the right value:

```
assertEquals("Shuang", p1.getName());
assertEquals(null, p1.getFather());
```

In the same way, we insert statements to test the getter methods of p2:

```
assertEquals("Lin", p2.getName());
assertEquals(p1, p2.getFather());
```

Running the test cases

Running the test cases is easy. First, compile the program and wait for button *Test* in the right half of the button bar of DrJava to become black. Then click it. The names of the procedures that begin with "test" appear in green in the bottom pane, and there is a green bar labeled "Test Progress". This means that executing calls on the test procedures detected no errors.

To show what happens when there is an error, let's change the expected value of the first call of `assertEquals`, compile again, and click button *Test*. Now, the name of the test procedure appears in red, meaning that an error was detected, and the offending call on `assertEquals` is highlighted.

We fix the error, compile, and test again. Now the tests work.

Testing the setter methods

We test the setter methods in the same way, putting the test cases —implemented in the `assertEquals` calls— in procedure `testSetters`. We write code to create one panda. And we write code to call each setter method and check whether it sets the field properly. Note that we make panda `p1` be its own father. It's ok, it's just for test purposes.

```
Panda p1= new Panda("Shuang", null);

p1.setName("Harry");
assertEquals("Harry", p1.getName());

p1.setFather(p1);
assertEquals(p1, p1.getFather());
```

Now compile and run the suite of test cases by clicking button *Test*. Voila! Everything is alright.

A few comments

A few comments are in order.

1. First, there is no need to put a specification on these test procedures, as long as the name of the procedure gives some indication of what is being tested.
2. Second, every method whose names begins with `test` will be called when button *Test* is clicked.
3. Third, note that there are many `assertEquals` procedures, one for each primitive type and one for each class type as well. You don't have to think about which one you are calling; just write the calls.
4. Fourth, in a call of `assertEquals`, the expected value is the first argument and the computed value, which should equal the expected value, is the second argument.
5. Fifth, how many different test procedures you write is up to you. Writing one test procedure for each method to be tested would be too much work. Don't do it! On the other hand, putting all test cases in one test procedure can make that test method too long and unwieldy. Don't do it!

   Instead, try to find a proper balance, by placing methods to be tested into logical groups —for example, getter methods, setter methods— and writing a test procedure for each group, so that no test procedure is too long and that they are all easily maintained.