

Testing assert statements

A method with a precondition may have assert statements to test that precondition, as in these examples:

```
/** Constructor: a P with grandparent p.  
 * Precondition: x is not null */  
public P(P p) {  
    assert p != null;  
    ...  
}
```

```
/** Change the name of the person to n.  
 * Precondition: n has at least one character. */  
public void changeName(String n) {  
    assert n != null && n.length() < 1;  
    ...  
}
```

Below is a way to test these assert statements, using a try-statement. You don't have to fully understand this if you haven't learned about exception handling yet. Just copy a try-statement given below and replace the red new-expression or method call in it by your appropriate call.

Testing an assert statement in a constructor

The following code placed in a testing procedure tests the assert statement in the constructor given above. The new-expression has an offending call on the constructor —its argument is null.

```
try {new P(null); fail("no exception thrown");}  
catch (AssertionError e) {if (e.getMessage() != null) fail();}
```

Case 1. The assert statement throws an AssertionError with a null detail message. The AssertionError is caught. Since e is null, the if-condition is false and the catch-block and thus the try-statement terminate normally. The assert statement was tested and it worked properly.

Case 2. Suppose the assert statement is not present in the constructor. Suppose the constructor call is executed without throwing an exception. Then the red `new P(null)`; is executed to completion and the following fail statement is executed. It throws an AssertionError with a non-null detail message. This is caught, and since e is non-null, the if-condition is true and the statement `fail()`; is executed. The try-statement terminates abnormally. It worked properly.

Case 3. Suppose the assert statement is not present and the constructor call throws some other exception. That exception is not caught by the catch-block and is thrown out further. Thus, the try-statement terminates abnormally. It worked properly.

Testing a call on a method

Now consider checking the assert statement in procedure `changeName`, shown to the right at the top of the page. Before we can call `changeName`, we need an object that contains that method. So, we use the following code. First, create an new P object and store it in p. Then, have two try-statements, the first to check `n != null` and the second to check `n.length() < 1`. That's it!

```
P p= new P(...); // Store in p an object that has method changeName
```

```
try {p.changeName(null); fail("no exception thrown");}  
catch (AssertionError e) {if (e.getMessage() != null) fail();}  
  
try {p.changeName(""); fail("no exception thrown");}  
catch (AssertionError e) {if (e.getMessage() != null) fail();}
```

A note on formatting

Generally, we would not scrunch up a try-statement onto to lines, the way we did above. We want a program to be as readable as possible. But this code has a certain structure, and only the stuff in red changes from test case to test case. We may have several of these in a testing procedure—even 5 or 6 or 10. In such a situation, scrunching the code up like this is preferred.