My First Unit Test

Dec / grocery

Dec2 / bank

Dec / gas

The test will have to

1. Load in test and target modules.
2. Create mock objects for the two subroutines to be called
3. Create a real object for the one to be tested.
4. Create the two mock subroutines with fixed returns
5. Call the real object/subroutine, feeding it a fixed value
6. Compare the results with an assert.

**THE TEST**

Load in modules:

1. Mockmodule-
2. Most –
3. Test –
4. Dec2, Dec – the modules involved.

use Test::MockModule;

use Test::Most;

use BERT::Test;

use Dec2;

use Dec;

Create two Mock Modules by passing the two target modules in. You will mock two subs inside these two mock modules.

 my $Dec2Mock = Test::MockModule->new('Dec2', no\_auto=>1);

my $Dec1Mock = Test::MockModule->new('Dec', no\_auto=>1);

The subroutine to be tested / invoked is now called:

my $dec=Dec->new;

Then: write the first subtest -

subtest "test grocery sub" => sub {

{#test 1 - feeding target sub (grocery) mock data

Mocks the two dependencies – give each values to return

$Dec2Mock->mock(bank => sub{

my $toReturn = 5;

return $toReturn;

});

$Dec1Mock->mock(gas => sub{

my $toReturn = 5;

return $toReturn;

});

Now, call the target, passing in a fixed number, then comparing the results with an expected return.

my $myinput = 10;

my $expected="36";

my $results = $dec->grocery($myinput);

assert ($results, $expected, "passed");

}

};

done\_testing();

**RUNNING THE TEST**

Here’s what the script runs:

Here’s what the test runs:

**More on the two modules tested**

I created two modules. The first, Dec2, begins with a constructor function ‘new’, which can be called by other modules that want to access the methods in this package:

package Dec2;

sub new {

my ($class) = @\_;

my $self = {};

bless $self, $class;

return $self;

}

Then it’s main sub, which simply returns $x:

sub bank {

$x=5;

return $x;

}

1;

The second module is called Dec.pm, and it begins by calling the module it will use (Dec2):

package Dec;

use Dec2;

After a constructor function (not shown, but identical to the one in the previous sub),

my $dec2 = Dec2->new;

we have

the main subroutine that will be tested. It does three things:

it gets $y by calling the gas sub

sub grocery {

($self, $args) = (@\_);

gas();

if (!$y) {

print "warning";

}

(2) it gets $x by calling the bank sub, which is invoked from the dec2 module.

$x=$dec2->bank();

if (!$x) {

print "warning";

}

3) it returns the sum of all these numbers, which will be **26:**

$a=$args+$x+$y+10;

print $a

return $a;

}

Finally, we write the gas sub itself, and begin the process by invoking grocery sub, which is our test-target:

sub gas {

$y=11;

return $y;

}

grocery();

1;

So ‘grocery’ is where the main stuff happens, and this calls on two subroutines in order to get extra details:

1. Gas is called from the same module
2. Bank is called from another module
3. It takes no inputs when as arguments. But it could if I called: grocery(12) and then began grocery sr with