SE465-001 Assignment 1

Kevin Carruthers (20463098), KevinJames@thekev.in

February 2, 2015

Question 1

I found the following results:

- in **C**: `-5%2` is -1
- in **Java**: `-5%2` is -1
- in **Perl**: `-5%2` is 1
- in **Python**: `-5%2` is 1

The reason for this discrepency is that C-based languages don't use modulus as an operator per-se, rather they use the remainder operator. Java and C, then, report negative values as a result.

Strategy 1. We can change the C, Java, etc compilers to return positive results. Something like

```
int modulus(int a, int b) {
    return a % b >= 0 ? a % b : (a % b) * -1;
}
```

Strategy 2. We could change the standards of what we expect a modulus to represent to include *equivalence classes*. By definition of modulus arithmetic, for `-5%2` we have -1 and 1 both in the same equivalence class as the answer (as would be $\ldots, -7, -5, -3, 3, 5, 7, \ldots$).

Question 2

Here are the requested test cases:

```
public class Question2Tests {

  @Test
  public void doesNotExecuteFault() {
    assertThrows(odd({}), NullPointerException());
  }
```

```
    public void executesFaultButNoErrorState() {
      assertEquals(odd({1, 2, 3, 4, 5}), 3);
    }

    public void errorButNoFailure() {
      // this is not possible, since any error state (incorrect value of {\tt count})
      // will be output (which makes it a failure as well as an error).
    }
  }
```

The first error occurs when an odd negative number (-9) is found in the list. The error state is a `count` which is one lower than it should be (ie. it is still 0 despite -9 being an odd number).

Question 3

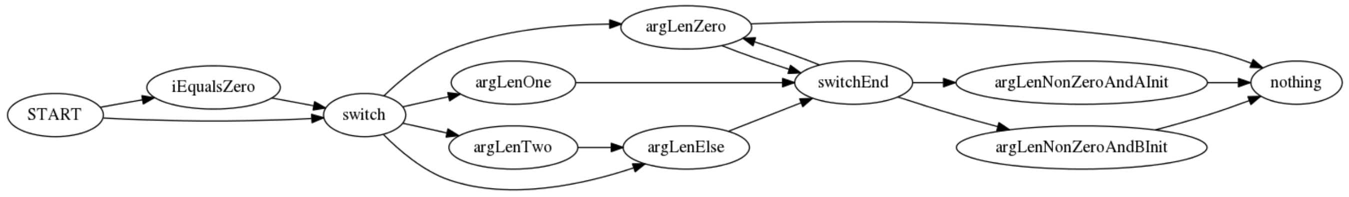

Figure 1: CFG for `M.m()`

(a)

(b) Test requirements for various forms of coverage:

- TR_{NC}: We must hit each node. This can be done by calling the function with an $i == 0$, an `arg` of length zero, an `arg` of length one, and an `arg` of length two or more.
- TR_{EC}: In this case, edge coverage is equivalent to node coverage (since there are no loops which edge coverage would require we run more than once). See the test cases for TR_{NC} to reach "full coverage".
- TR_{EPC}: Edge-pair coverage requires we visit all edges of length two in our program. Since this is impossible – because we can not follow paths which require `arg.length()` to be zero to reach the first edge-pair and non-zero for the second – we can only attempt to "do our best": see the test cases for TR_{EC} to get as close as possible.
- TR_{PPC}: Prime Path coverage requires we cover all paths which are not proper subsets of other paths. This, again, can not give us "perfect coverage" for the same reasons as outlined in TR_{EPC}. Again, those test requiremments get as as close to prime-path coverage as possible.

It is impossible to satisfy any test requirements since each of them require we visit or pass through a node which can only be reached if `argv.length = 0`. Since our main function prohibits this, we can not acheive "full coverage".

Question 4

Part 1: Test Requirements

I will test the functions `calculate_maxima()` and `calculate_raw_allocations()`.

`calculate_maxima()`. Test requirements:

- All of: no accounts and no holdings, no accounts and holdings, accounts and no holdings, accounts and holdings
- Both of: accounts with `a.can_add_money == True` and `a.can_add_money == False`

`calculate_raw_allocations()`. Test requirements:

- A test case with no allocation rules
- A test case with a single allocation rules
- A test case with several equivalent allocation rules
- A test case with several different allocation rules

Part 2: Implementation and Coverage Results

The coverage report from my tests is below:

```
Name                              Stmts   Miss  Cover   Missing
---------------------------------------------------------------
track                                 0      0   100%
track.migrations                      0      0   100%
track.migrations.0001_initial         5      0   100%
track.models                         45     41     9%   1-20, 23-26, 29-40,
                                                        43-48, 51-57
track.views                          85     29    66%   74-89, 107-126
---------------------------------------------------------------
TOTAL                               135     70    52%
---------------------------------------------------------------
```

Part 3: Fixing a bug; statement coverage

One of the bugs in `distribute` is `calculate_maxima`; this function returns `'N/A'` when `.can_add_money` is `True` as opposed to the opposite, which would be expected. This is shown by the test case `testCalculateMaximai()`.

Before the fix:

```
======================================================================
FAIL: testCalculateMaxima (track.tests.ViewsTestCase)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/kevin/coding/stqam/a1/q4/distribute/track/tests.py",
      line 57, in testCalculateMaxima
```

```
    Account.objects.get(name='acc3'): 345,
AssertionError: Tuples differ:
    ({<Account: acc1 (RRSP)>: (Dec... !=
    ({<Account: acc1 (RRSP)>: (123...

First differing element 0:
{<Account: acc1 (RRSP)>: (Decimal('123'), Decimal('123')),
 <Account: acc2 (TFSA)>: (Decimal('666'), 'N/A'),
 <Account: acc3 (NR)>: (Decimal('345'), 'N/A')}
{<Account: acc1 (RRSP)>: (123, 'N/A'),
 <Account: acc2 (TFSA)>: (666, 666),
 <Account: acc3 (NR)>: (345, 345)}
```

After the fix:

```
OK
```

I changed line 25 from `if a.can_add_money` to `if not a.can_add_money`.

This change does not improve test coverage since the lines are executed either way, they simply return incorrect output.