Inf2-SEPP 2024-25

Tutorial 6 (Week 10)

Notes on answers

1 Task 1: Unit Testing and Test Coverage

1. An example of an assert statement is:

   assertEquals(15, calculator.getReducedCalories(85,true,false), "Correct initial calories with semi skimmed milk should compute correctly");

   Values in the different possible intervals and at the boundaries are selected for initialCalories, with the intervals being different depending on the replacement contents selected:

   - any value for initialCalories if no replacement contents selected; the result is -1
   - less than 79, 79, 79-250, 250, greater than 250 if only semi-skimmed milk selected; result is -1 for less than 79 or greater than 250, and should be computed correctly for the others
   - less than 89, 89, 89-250, 250, greater than 250 if only sugar free syrup selected; result is -1 for less than 89 or greater than 250, and should be computed correctly for the others
   - less than 159, 159, 159-250, 250, greater than 250 if both selected; result is -1 for less than 159 or greater than 250, and should be computed correctly for the others
   - It is also useful to take values right outside the correct ranges to test that the method would return -1.

   Therefore, the following are the combinations of inputs and expected results, where each should be placed in a different test for thorough testing:

   - Test1: Any value for initialCalories and no replacements selected: 70, false, false; result should be -1
   - Incorrect value for initialCalories in combination with:

  - Test2: skimmed milk, with initialCalories too small (it should be minimum 79) but right outside range, with skimmed milk only selected: 78, true, false; result should be -1
  - Test3: skimmed milk, with initialCalories too small (it should be minimum 79), with skimmed milk only selected: 60, true, false; result should be -1
  - Test4: skimmed milk, with initialCalories too large (it should be maximum 250) but right outside range, with skimmed milk only selected: 251, true, false; result should be -1
  - Test5: skimmed milk, with initialCalories too large (it should be maximum 250), with skimmed milk only selected: 270, true, false; result should be -1
  - Test6: sugar free syrup, with initialCalories too small (it should be minimum 89) but right outside range, with sugar free syrup only selected: 88, false, true; result should be -1
  - Test7: sugar free syrup, with initialCalories too small (it should be minimum 89), with sugar free syrup only selected: 72, false, true; result should be -1
  - Test8: sugar free syrup, with initialCalories too large (it should be maximum 250) but right outside range, with sugar free syrup only selected: 251, false, true; result should be -1
  - Test9: sugar free syrup, with initialCalories too large (it should be maximum 250), with sugar free syrup only selected: 263, false, true; result should be -1
  - Test10: both skimmed milk and sugar free syrup, with initialCalories too small (it should be minimum 159) but right outside range, with both skimmed milk and sugar free syrup selected: 158, true, true; result should be -1
  - Test11: both skimmed milk and sugar free syrup, with initialCalories too small (it should be minimum 159), with both skimmed milk and sugar free syrup selected: 130, true, true; result should be -1
  - Test12: both skimmed milk and sugar free syrup, with initialCalories too large (it should be maximum 250) but right outside range, with both skimmed milk and sugar free syrup selected: 251, true, true; result should be -1
  - Test13: both skimmed milk and sugar free syrup, with initialCalories too large (it should be maximum 250), with both skimmed milk and sugar free syrup selected: 288, true, true; result should be -1
- Correct but marginal value for initialCalories in cobination with:
  - Test14: skimmed milk syrup, with skimmed milk only selected (low initialCalories marginal value): 79, true, false; result should be 9
  - Test15: skimmed milk syrup, with skimmed milk only selected (high initialCalories marginal value): 250, true, false; result should be 180
  - Test16: sugar free syrup, with sugar free syrup only selected (low initialCalories marginal value): 89, false, true; result should be 9
  - Test17: sugar free syrup, with sugar free syrup only selected (high initialCalories marginal value): 250, false, true; result should be 170
  - Test18: Both skimmed milk and sugar free syrup, with both skimmed milk and sugar free syrup selected (low initialCalories marginal value): 159, true, true; result should be 9

  - Test19: Both skimmed milk and sugar free syrup, with both skimmed milk and sugar free syrup selected (high initialCalories marginal value): 250, true, true; result should be 100
- Correct non-marginal value for initialCalories in cobination with:
  - Test20: skimmed milk, with skimmed milk only selected: 85, true, false; result should be 15
  - Test21: sugar free syrup, with sugar free syrup only selected: 97, false, true; result should be 17
  - Test22: both skimmed milk and sugar free syrup, with both skimmed milk and sugar free syrup selected: 170, true, true; result should be 20

2. An example of how a test would look like

```
@Test
@DisplayName("Testing correct value for initialCalories with skimmed
milk only selected")
public void testSkimmedOnly() {
    assertEquals(15, calculator.getReducedCalories(85,true,false),
    "Correct initial calories with semi skimmed milk should compute correctly");
}
```

   In total there should be 22 tests in the test suite, as shown in point 1.

3. We have 8 statements in our method's code (the closing bracket is not usually counted as a separate statement). The 'return -1;' statement on line 3 is not covered by any test, while all the other statements are, so our tests cover 7 out of the 8 statements. Therefore, statement coverage is 7 / 8 = 87.5%.

   We have 6 branches in our code, corresponding to the 3 if conditions being true or false. Our tests only cover 5 of them, as the first condition is never true for our tests. Therefore, branch coverage is 5 / 6 = 83.33%.

4. The statement on line 3 is never reachable because the condition can never be true. Therefore, 87.5% is the maximum statement coverage we can get with this code. Moreover, the maximum branch coverage we can reach is 83.33%.

   In conclusion, for our chosen tests we have reached the maximum possible statement and branch coverage. If your tests were different and you missed some of the possible classes of inputs for step 1, you may not have reached them, and additional tests would help reach them. Write and run these tests.

5. Test 1 will fail because case with no replacement contents selected returns initialCalories in the code, and not -1 as required.

   Tests 2-13 will fail because the first condition in the code (which was intended to check for calory values outside range) can never be met (`initialCalories` cannot be both less than 9 and greater than 250) and so the code will never return -1. The code does not consider how the calory interval is affected by the selected replaced contents, or that both replacement contents can be selected. Therefore, for the inputs provided in the tests the method will return values outside range or incorrect.

Tests 18, 19, 22 will fail because the cases with both skimmed milk and sugar free syrup selected are not considered.

In conclusion, only tests 14-17, 20, 21 will pass.

The code can be improved by:

- removing the previous first (incorrectly written) condition
- adding a condition at the top (most efficient) for the case where no replacement contents are requested, returning -1
- adding conditions to take remaining cases of `skimmedMilk` and `sugarFreeSyrup` value combinations; within each, adding further conditions to check for initialCalories belonging to the correct intervals based on the replaced component(s); if not, returning -1; if so, making all the necessary deductions before returning

Here is a way of solving this:

```
int getReducedCalories(int initialCalories, boolean skimmedMilk,
boolean sugarFreeSyrup){
     if ((skimmedMilk==false) && (sugarFreeSyrup==false))
          return -1;
     if ((skimmedMilk==true) && (sugarFreeSyrup==false)){
          if ((initialCalories<79) || (initialCalories>250))
               return -1;
          else
               return initialCalories - 70;
    }
    else
          if ((skimmedMilk==false) && (sugarFreeSyrup==true)){
               if ((initialCalories<89) || (initialCalories>250))
                    return -1;
               else
                    return initialCalories - 80;
          }
          else{ //only case remaining is that in which both are true
                if ((initialCalories<159) || (initialCalories>250))
                    return -1;
               else
                    return initialCalories - 150;
          }
}
```

To check that the code is now correct, the tests should be run again and should this time all pass.

2 Task 2: System Testing

1. To support the given use case, the addCoffee method in the OrderManager class would be first called (once if one cofee is to be ordered, or repeatedly if several coffees are to be ordered as per extension 3a). This method would first check the validity of the inputs (see extension 2a) and return a String "Invalid inputs provided" if the quanity or coffee type are incorrect. It would then create coffees (objects of class Coffee) using constructors of the right coffee type (the OrderManager would know which coffee type class each coffee type corresponds to) and add them to the order manager's collection of coffees. Finally, for each successfully added coffee it would return the String confirming the coffee type with quantity was requested.

   Then, the placeOrder method would be called. It would first retrieve each of the needed ingredients (coffeeBeanQuantity, milkQuantity) required by each coffee in its newly made collection by using the getters from Coffee (omitted in diagram). Then, it would iterate over its collection of objects of type Ingredient to identify the associated ingredient of the type needed for the coffee. It would check if its stock can accommodate the needed quantity (by calling its enoughStock method). If for any of the coffees and any of the ingredients there is no stock left, the placeOrder method would return the String "Not enough stock to make one or more coffees" (extensions 4a and 4b). If there is enough stock for each coffee and all of the ingredients, it would iterate over the coffees and ingredients again and for each ingredient decrease its quantity as needed by the coffee (using its decreaseStock method). Next, placeOrder would create a new object of class Order holding the needed coffees. Finally, it would call the order's makeFullStatement method. This method would call each of the the order's private methods in turn, which would query the coffees for their caffeine, fat, calories and price (the methods in the right coffee type classes used at runtime through polymorphism), make any needed sums (extension 4c) and produce that part of the statement. Finally, makeFullStatement puts the statement together, also including the order id, in a String to return. placeOrder gets this String and returns it in turn.

   We assume a different OrderManager object is created for each customer's orders.

2. You are encouraged to write this code and get it to work, also considering guidelines for high quality code, as this is excellent practice of programming.

3. It is advisable to write a @BeforeAll method to set up the test environment first, consisting of an object of the OrderManager class and objects of the Ingredient class setting up the types of ingredients and stock levels for each. Then, tests should take as many as possible of the following scenarios: Main success scenario, scenario 1-2a, scenario 1-2-3a-1-2a, scenario 1-2-3-4a, scenario, 1-2-3a-1-2-3-4b, scenario 1-2-3a-1-2-3-4c.

   One test is enough for each of the above, but several tests can be added for each with different types of coffee, different quantities, and with different combinations of normal and skimmed milk, for maximum coverage.

   You are encouraged to write the tests, run coverage (statement and branch) on your IDE and strive to improve coverage by adding more tests.

Cristina Adriana Alexandru. 20 March 2025.