

Technical Report for Coffee Maker Program

Allen Bates, Hao Chen, Marcos Moraes, Krishna Sindhuja

Defects Found:

Main.java

- In `recipeListSelection(String)` if the recipe selection is out of bounds, an exception is thrown and the system crashes. A message should be printed telling users to select a number from 1-3. Can be solved by recursively calling the method. (Non-equivalent mutant)

```
recipe = recipeListSelection("Please select a number from 1-3.");
```

- If the recipe doesn't exist, we should check if `recipeForPurchase == null`, and if it is null it should skip the next step or say select from 1-3

Recipe.java

- `setName(String)` needs to check for blank names. i.e. Empty Strings

RecipeBook.java

- `editRecipe(int, Recipe)` deletes Recipe's name
- `deleteRecipe(int)` deletes the recipe but doesn't remove a recipe item from the `recipeArray`. Instead of setting `recipeArray[int]` to null, it sets it to a new instance of the Recipe class
- `private final int NUM_RECIPES = 4;` Should be 3
- Each recipe must have a unique name
- Check if `recipeArray` is empty in `editRecipe(int)` and `deleteRecipe(int)`. If `recipeArray` is empty, continue to calling `mainMenu()`
- Check `recipename` for null.

CoffeeMaker.java

Inventory.java

- `useIngredients(Recipe)` increments the inventory's amount of coffee instead of decrementing it
- `addSugar(String)` should be checking if the parameter 'sugar' is greater than or equal to zero, not less than or equal to zero.

Test Cases

RecipeTest.java

First initialize 2 new recipe r1 and r2. Their names are blank strings, and all the data are int 0. getAmtChocolateTest() calls assertEquals() to check if the amtChocolate of r1 and r2 are the same.

setAmtChocolateTest() calls r1.setAmtChocolate("1") and calls assertEquals() to check if the amtChocolate of r1 has changed to 1. Then do the same thing but set r1.amtChocolate to 0 to test boundary in setAmtChocolateTest() hasn't changed. Then test inputs (-1, a) that should throw exceptions. Catch the exception and compare it with the expected output.

getAmtCoffeeTest() calls assertEquals() to check if the amtCoffee of r1 and r2 are the same.

setAmtCoffeeTest() calls r1.setAmtCoffee("1") and calls assertEquals() to check if the amtCoffee of r1 has changed to 1. Then do the same thing but set r1.amtCoffee to 0 to test boundary in setAmtCoffeeTest() hasn't changed. Then test inputs (-1, a) that should throw exceptions. Catch the exception and compare it with the expected output.

getAmtMilkTest() calls assertEquals() to check if the amtMilk of r1 and r2 are the same.

setAmtMilkTest() calls r1.setAmtMilk("1") and calls assertEquals() to check if the amtMilk of r1 has changed to 1. Then do the same thing but set r1.amtMilk to 0 to test boundary in setAmtMilkTest() hasn't changed. Then test inputs (-1, a) that should throw exceptions. Catch the exception and compare it with the expected output.

getAmtSugarTest() calls assertEquals() to check if the amtSugar of r1 and r2 are the same.

setAmtSugarTest() calls r1.setAmtSugar("1") and calls assertEquals() to check if the amtSugar of r1 has changed to 1. Then do the same thing but set r1.amtSugar to 0 to test boundary in setAmtSugarTest() hasn't changed. Then test inputs (-1, a) that should throw exceptions. Catch the exception and compare it with the expected output.

getNameTest() calls assertTrue() to check if the name of r1 is as expected blank string.

setNameTest() calls r1.setName(null) to check the name should not be changed. Then calls r1.setName("2333"), and calls assertEquals() to check if the name is changed to "2333".

getPriceTest() calls assertEquals() to check if the price of r1 and r2 are the same.

setPriceTest() calls r1.setPrice("1") and calls assertEquals() to check if the price of r1 has changed to 1. Then do the same thing but set r1.price to 0 to test boundary in setPriceTest() hasn't changed. Then test inputs (-1, a) that should throw exceptions. Catch the exception and compare it with the expected output.

toStringTest() calls r1.setName("1") to change r1.name, and call assertEquals() to compare r1.toString() with the expected value "1".

hashCodeTest() recipe hashCode is based on name.hashCode(), so call assertEquals() to compare r1.hashCode() with r2.hashCode(), they should be the same.

equalsTest() calls assertTrue() to check r1 equals to r1. Calls assertFalse() to check r1 doesn't equals to null. Calls assertFalse() to check r1 doesn't equals r1.getAmtChocolate() because it's not the same object. Calls assertTrue(), r1 and r2 has the same initial name, the method would return true if names of two objects are the same. Change r2.name, calls assertFalse() to check r1 is not equal to r2 now.

CoffeeMakerTest.java

testAddRecipe() initializes a new CoffeeMaker object. assertTrue() is called on the return value of the CoffeeMaker object calling addRecipe()' with an initialized and defined recipe as it's argument.

testDeleteRecipe() initializes a new CoffeeMaker object. After the object adds a recipe, assertEquals() is called with the name of the recipe as it's expected output and the return value of the object calling deleteRecipe(), with zero as its argument, as it's tested output.

testEditRecipe() initializes a new CoffeeMaker object. After the object adds a recipe, assertEquals() is called with the name of a new recipe as it's expected output and the return value of the object calling editRecipe(), replacing the original recipe with the new recipe, as it's tested output.

testAddInventory() initializes a new CoffeeMaker object, then calls addInventory(). A new StringBuffer instance is created. The buffer appends the appropriate strings that are respective to what checkInventory() should return, at that moment. assertEquals() is called with the buffer, as a string, as the expected output and the return value of the CoffeeMaker object calling checkInventory() as the tested output. **Note: Originally, this test will not work because the addSugar() function checks if the argument of the function is less than or equal to zero. It should be checking if the argument is greater than or equal to zero.**

testCheckInventory() initializes a new CoffeeMaker object. A new StringBuffer instance is created. The buffer appends the appropriate strings that are respective to what checkInventory() should return, initially. assertEquals() is called with the buffer, as a string, as the expected output and the return value of the CoffeeMaker object calling checkInventory() as the tested output.

testMakeCoffeeNoRecipe() initializes a new CoffeeMaker object. assertEquals() is then called with the return value of the object calling makeCoffee() as the tested output. The expected output is set to the same value of the second argument of makeCoffee(). Since no recipes have been added, the value of the second argument of makeCoffee() should be returned.

testMakeCoffeeEnoughIngredients() initializes a new CoffeeMaker object. The object adds a recipe that requires a smaller or equal amount of ingredients compared to the initial amount of ingredients in the inventory. assertEquals() is then called with the return value of the object calling makeCoffee() as the tested output. The second argument of makeCoffee() will be set to a value higher than the price value of the added recipe. The expected output is set to the value of the second argument of makeCoffee() minus the price of the added recipe.

testMakeCoffeeNotEnoughIngredients() initializes a new CoffeeMaker object. The object adds a recipe that requires a larger amount of ingredients compared to the initial amount of ingredients in the inventory. assertEquals() is then called with the return value of the object calling makeCoffee() as the tested output. The expected output is set to the value of the second argument of makeCoffee().

testMakeCoffeeNotEnoughCash() initializes a new CoffeeMaker object. The object adds a recipe. assertEquals() is then called with the return value of the object calling makeCoffee() as the tested output. The second argument of makeCoffee() will be set to a value lower than the price value of the added recipe. The expected output is set to the value of the second argument of makeCoffee().

testGetRecipes() declares and initializes a new array of Recipe objects. The array is initialized with four recipe objects. A new CoffeeMaker object is initialized and it calls addRecipe() four times, each time adding the recipe that was used to initialize the Recipe array. The recipes are added in order so that the CoffeeMaker object's Recipe array is identical to the test case's Recipe array. assertEquals() is called with the Recipe array as the expected output and the return value of the CoffeeMaker object calling getRecipes() as the tested output.

InventoryTest.java

addChocolateErrorTest() - Creates a local String variable qty with value of "a". Later it gets the current quantity of chocolate and calls the add method but gives an exception because "a" is not a number.

addChocolateTest() - Creates a local String variable qty with value of "5". Later it gets the current quantity of chocolate and calls the add method to add "5" to the object inventory and assertEquals the sum of the inventory before the add plus the amount added to the current chocolate inventory.

addCoffeeErrorTest() - Creates a local String variable qty with value of "w242s". Then it gets the current quantity of coffee with the getCoffee() and calls the addCoffee() method but gives an InventoryException because "w242s" is not a number.

addCoffeeTest() - Creates a local String variable qty with value of "5", gets the current quantity of coffee with the getCoffee() and calls the addCoffee() method to add "5" to the inventory object and assertEquals the sum of the inventory before the add plus the amount added to the current coffee inventory.

addMilkErrorTest() - Creates a local String variable qty with value of "X". Then it gets the current quantity of milk with the getMilk() and calls the addMilk() method but gives an InventoryException because "X" is not a number.

addMilkTest() - Creates a local String variable qty with value of "5", gets the current quantity of milk with the getMilk() and calls the addMilk() method to add "5" to the inventory object and assertEquals the sum of the inventory before the add plus the amount added to the current milk inventory. The fail command is not called because the assertEquals runs correctly.

addSugarErrorTest() - Creates a local String variable qty with value of "12s". Then it gets the current quantity of sugar with the getSugar() and calls the addSugar() method but gives an InventoryException because "12s" is not a number.

addSugarTest() - Creates a local String variable qty with value of "2", gets the current quantity of msugar with the getSugar() and calls the addSugar() method to add "2" to the inventory object and assertEquals the sum of the inventory before the add plus the amount added to the current sugar inventory.

enoughIngredientsErrorTest() - First a Recipe object is created. It uses the default constructor then set amount for coffee, milk, sugar and chocolate. The exception is thrown when it tries to assert true with the method enoughIngredients() and it returns false because there is not enough ingredients to make this recipe.

enoughIngredientsTest() - First a Recipe object is created. It uses the default constructor then set amount for coffee, milk, sugar and chocolate. After setting the ingredients it assertTrue with the method enoughIngredients() for this recipe.

getChocolateTest() - It has 1 assertEquals to test the correct quantity of chocolate.

getCoffeeErrorTest() - It asserts a different quantity of coffee from the current quantity of the inventory object.

getChocolateErrorTest() - It has 1 assertEquals to test the wrong quantity of chocolate. This test is not completed fully because the assertEquals throws an error because the quantity is different from the object.

getCoffeeErrorTest() - It asserts a different quantity of coffee from the current quantity of the inventory object.

getCoffeeTest() - It asserts the correct quantity of coffee from the current quantity of the inventory object.

getMilkErrorTest() - It asserts a different quantity of milk from the current quantity of the inventory object.

getMilkTest() - It asserts the correct quantity of milk from the current quantity of the inventory object.

getSugarErrorTest() - It asserts a different quantity of sugar from the current quantity of the inventory object.

getSugarTest() - It asserts the correct quantity of sugar from the current quantity of the inventory object.

setChocolateTest() - First this method sets the chocolate quantity to 10 to the inventory object and assertsEquals to the correct quantity. Then it assertsEquals to a different quantity from the inventory object.

setCoffeeErrorTest() - This method first sets the coffee quantity to 11, then assertsEquals to a different quantity of coffee.

setCoffeeTest() - First sets the quantity of coffee to 10 and then asserts to the correct quantity of the inventory object.

setMilkErrorTest() - This method first sets the milk quantity to 23, then assertsEquals to a different quantity of milk.

setMilkTest() - First sets the quantity of mile to 10 and then asserts to the correct quantity of the inventory object.

setSugarErrorTest() - This method first sets the sugar quantity to 99, then assertsEquals to a different quantity of sugar.

setSugarTest() - First sets the quantity of mile to 10 and then asserts to the correct quantity of the inventory object.

setUp() - Creates a new Inventory object and this method runs before others. It has the annotation @before.

toStringTest() - assertsNotNull the return of the toString method in the inventory class.

useIngredientsErrorTest() - First a recipe object is created and set coffee, milk, sugar, chocolate quantity. After the sets assertTrue if the inventory class returns true. The exception is thrown because a negative number is passed by parameter.

useIngredientsTest() - First a recipe object is created and set coffee, milk, sugar, chocolate quantity. After the sets assertTrue if the inventory class returns true.

RecipeBookTest.java:

Creates a recipeBook() object and four recipe() objects.

Sets up the values for the four recipe objects.

r1 and r2 are made to add to recipe and then called in testGetRecipes(). The test compares whether the added recipe and the called recipe object are referring to the same array location and array value.

Since r1 is added to the recipebook object for test already, adding r1 again should throw an error. testAddRecipeFalse() identifies that.

Since r3 is not added to the recipebook object for test already, adding r3 run smoothly.

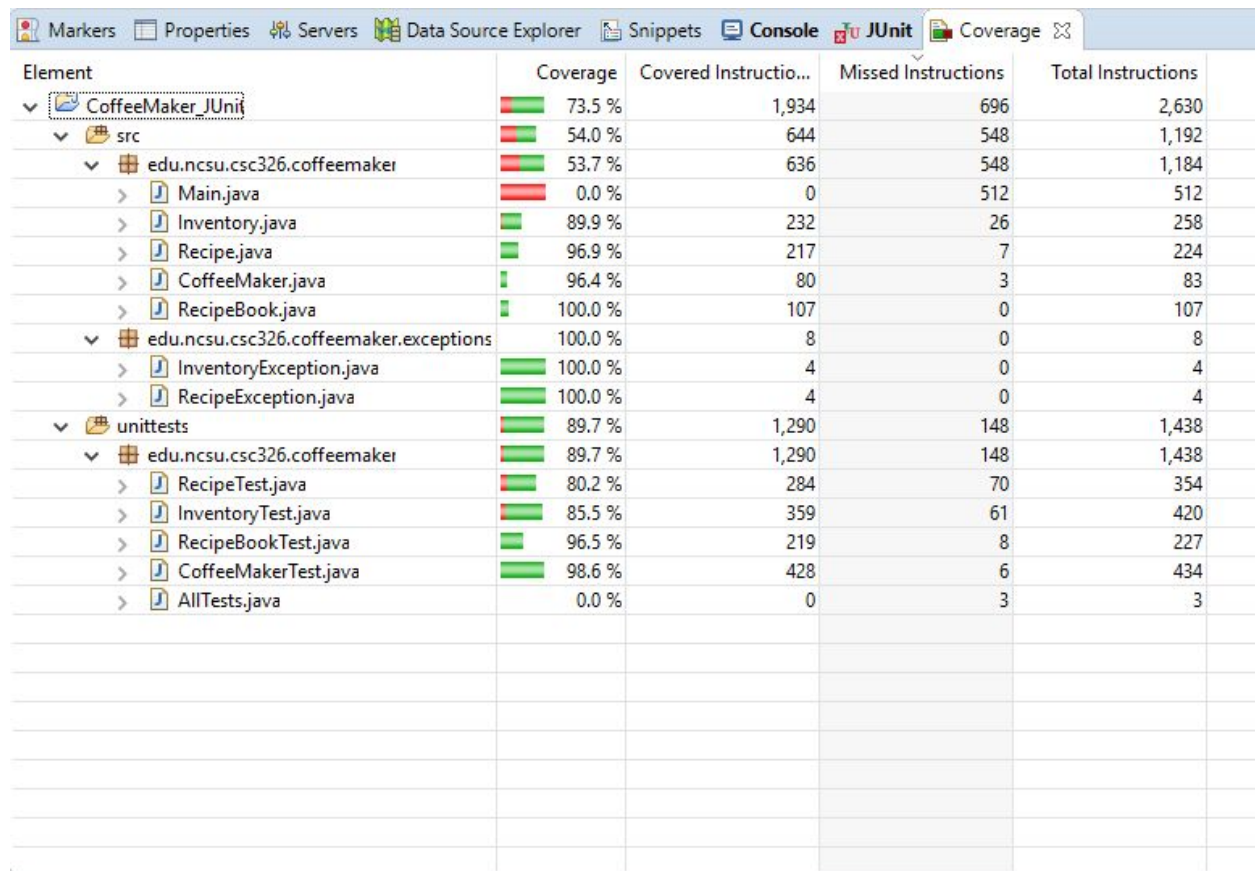
testAddRecipeTrue() tests for that.

testDeleteRecipeName() deletes the entry the that is there on the recipeBook().

testDeleteRecipeNull() identifies whether the entry deleted is null or not and if the entry is deleted completely.

testEditRecipeName() checks whether the edited recipe the name of the recipe.

Then testEditRecipeNull() checks if the edited recipe is null.



Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
▼ CoffeeMaker_JUnit	73.5 %	1,934	696	2,630
▼ src	54.0 %	644	548	1,192
▼ edu.ncsu.csc326.coffeemaker	53.7 %	636	548	1,184
> Main.java	0.0 %	0	512	512
> Inventory.java	89.9 %	232	26	258
> Recipe.java	96.9 %	217	7	224
> CoffeeMaker.java	96.4 %	80	3	83
> RecipeBook.java	100.0 %	107	0	107
▼ edu.ncsu.csc326.coffeemaker.exceptions	100.0 %	8	0	8
> InventoryException.java	100.0 %	4	0	4
> RecipeException.java	100.0 %	4	0	4
▼ unittests	89.7 %	1,290	148	1,438
▼ edu.ncsu.csc326.coffeemaker	89.7 %	1,290	148	1,438
> RecipeTest.java	80.2 %	284	70	354
> InventoryTest.java	85.5 %	359	61	420
> RecipeBookTest.java	96.5 %	219	8	227
> CoffeeMakerTest.java	98.6 %	428	6	434
> AllTests.java	0.0 %	0	3	3

Figure 1.1: Emma Coverage of Test Suite