

## Lab (5)



**Program: Computer and  
Systems Engineering**

***Course Code: CSE337s***

***Course Name: Software Testing***

**Ain Shams University  
Faculty of Engineering**





**Team Number: 7**

**TEAM MEMBERS:**

Name	ID	Level
Mazen Ehab Mohamed Maher	1901120	Senior 2
Ahmed Mahmoud Mohamed Ibrahim	1901143	Senior 2
Mohamed Mostafa Shaban Mohamed	1901650	Senior 2
Mostafa Mohamed Ahmed Abdelaal	1803093	Senior 2
Andrew Samir Kamel Gayed	1900242	Senior 2
AbdAllah Mostafa Mahmoud Alsayed	1900779	Senior 2




## Contents

First: Refactor the code by splitting it into modules.....	4
1- <i>CoffeeMachineRefactored</i> .....	4
2- <i>CoffeeMachineUI</i> .....	6
Second: Use Junit to apply integration testing by.....	8
1- Make unit testing for <i>CoffeeMachineRefactored</i> class.....	8
The code and test cases of the <i>CoffeeMachineRefactoredTester</i> class	8
2- Top-Down Approach .....	22
The code and test cases of <i>CoffeeMachineUITopDownTester</i> class .....	23
3- Bottom-Up Approach .....	35
The code and test cases of <i>CoffeeMachineUIBottomUpTester</i> class. ..	36
Link of Source Code .....	53

## First: Refactor the code by splitting it into modules

The code is split into two classes called (  CoffeeMachineRefactored.java ,

 CoffeeMachineUI.java )

### 1-CoffeeMachineRefactored

```
*CoffeeMachineRefactored.java X
1 package CoffeeMachine;
2
3 public class CoffeeMachineRefactored
4 {
5     /***** attributes *****/
6     private double coffee_powder;
7     private double milk;
8     private double water;
9     private int Coffee_Count;
10    /***** constructor *****/
11    public CoffeeMachineRefactored(double coffee_powder, double milk, double water) { // Constructor Initialization...
12        this.coffee_powder = coffee_powder;
13        this.milk = milk;
14        this.water = water;
15        this.Coffee_Count = 0;
16    }
17    /***** getters *****/
18    public double getPowder() {return coffee_powder;}
19    public double getMilk() {return milk;}
20    public double getWater() {return water;}
21    public int getCoffee_Count() {return Coffee_Count;}
22
23    /***** setters *****/
24    public void setPowder(double coffee_powder) {this.coffee_powder = coffee_powder;}
25    public void setMilk(double milk) {this.milk = milk;}
26    public void setWater(double water) {this.water = water;}
27
28    /***** fundamental functions *****/
29    public void SetIngredient(){ //Calling for Filling Ingredient...
30        this.coffee_powder = 500.0;
31        this.milk = 1;
32        this.water= 2;
33    }
34    public void CleanMachine(){ //Initialization with Null In order to Clean Machine
35        this.coffee_powder = 0;
36        this.milk = 0;
37        this.water= 0;
38    }
39    public int makecoffee(char t){ //Coffee Selection Menu
40        int result = 0;
41        boolean available = false;
42        switch(t)
43        {
44            case '1':
45                available = this.BlackCoffee(); //Call to BlackCoffee Method
46                if(available)
47                {
```



```
"CoffeeMachineRefactored.java"
44     case '1':
45         available = this.BlackCoffee();    //Call to BlackCoffee Method
46         if(available)
47         {
48             result = 1;
49         }
50         else
51         {
52             result = 0;
53         }
54         break;
55     case '2':
56         available = this.MilkCoffee();    //Call to MilkCoffee Method
57         if(available)
58         {
59             result = 2;
60         }
61         else
62         {
63             result = 0;
64         }
65         break;
66     case '0':
67         result = 0;
68         break;
69     default:
70         result = t - '0';
71         break;
72 }
73 return result;
74 }
75 public boolean BlackCoffee(){
76     if(this.coffee_powder >= 10 && this.water >= 0.2){
77         this.coffee_powder -= 10;
78         this.water -= 0.2;
79         this.Coffee_Count++;
80         return true;
81     }
82     return false;
83 }
84 public boolean MilkCoffee(){
85     if(this.coffee_powder >= 10 && this.milk >= 0.4 && this.water >= 0.2){
86         this.coffee_powder = this.coffee_powder - 10;
87         this.milk -= 0.4;
88         this.water -= 0.2;
89         this.Coffee_Count++;
90         return true;
91     }
92     return false;
93 }
94 }
95 }
```

## 2-CoffeeMachineUI

```
*CoffeeMachineRefactored.java *CoffeeMachineUI.java X
1 package CoffeeMachine;
2
3 import java.util.Scanner;
4 public class CoffeeMachineUI
5 {
6     /***** attributes *****/
7     static Scanner scan;
8     CoffeeMachineRefactored CoffeeMachineRefactoredObject;
9
10    /***** constructor *****/
11    public CoffeeMachineUI(CoffeeMachineRefactored c)
12    {
13        scan = new Scanner(System.in);
14        this.CoffeeMachineRefactoredObject = c;
15    }
16
17    /***** fundamental functions *****/
18    private void GetIngredient(){ //To Get Status
19        System.out.println("Available Coffee Power(Gram) "+String.format("%.1f",CoffeeMachineRefactoredObject.getPowder()));
20        System.out.println("Available Milk(Liter) "+String.format("%.1f", CoffeeMachineRefactoredObject.getMilk()));
21        System.out.println("Available Water(Liter) "+String.format("%.1f", CoffeeMachineRefactoredObject.getWater()));
22    }
23
24    public void start()
25    { //public Start to access all private method of this class
26        System.out.println(" -----");
27        System.out.println("| Coffee Machine By Manikant |");
28        System.out.println(" -----");
29
30        System.out.println("\nCurrent Status: ");
31
32        this.GetIngredient();
33
34        boolean t = true;
35        while(t){
36            System.out.println("\n ----- ");
37            System.out.println("|1: Status of Ingredient | \n ----- \n|2: Fill Ingredient");
38            System.out.println(" ----- \n\n");
39            char c = CoffeeMachineUI.scan.next().charAt(0);
40            switch(c){
41                case '1':
42                    System.out.println("----- Status -----");
43                    this.GetIngredient();
44                    System.out.println("-----");
45                    break;
46                case '2':
47                    System.out.println("\nFilling...");
48                    CoffeeMachineRefactoredObject.SetIngredient();
49                    System.out.println("Filling Completed.");
50                    break;
51                case '3':
52                    System.out.println("\nCleaning Machine...");
53            }
54        }
55    }
56 }
```

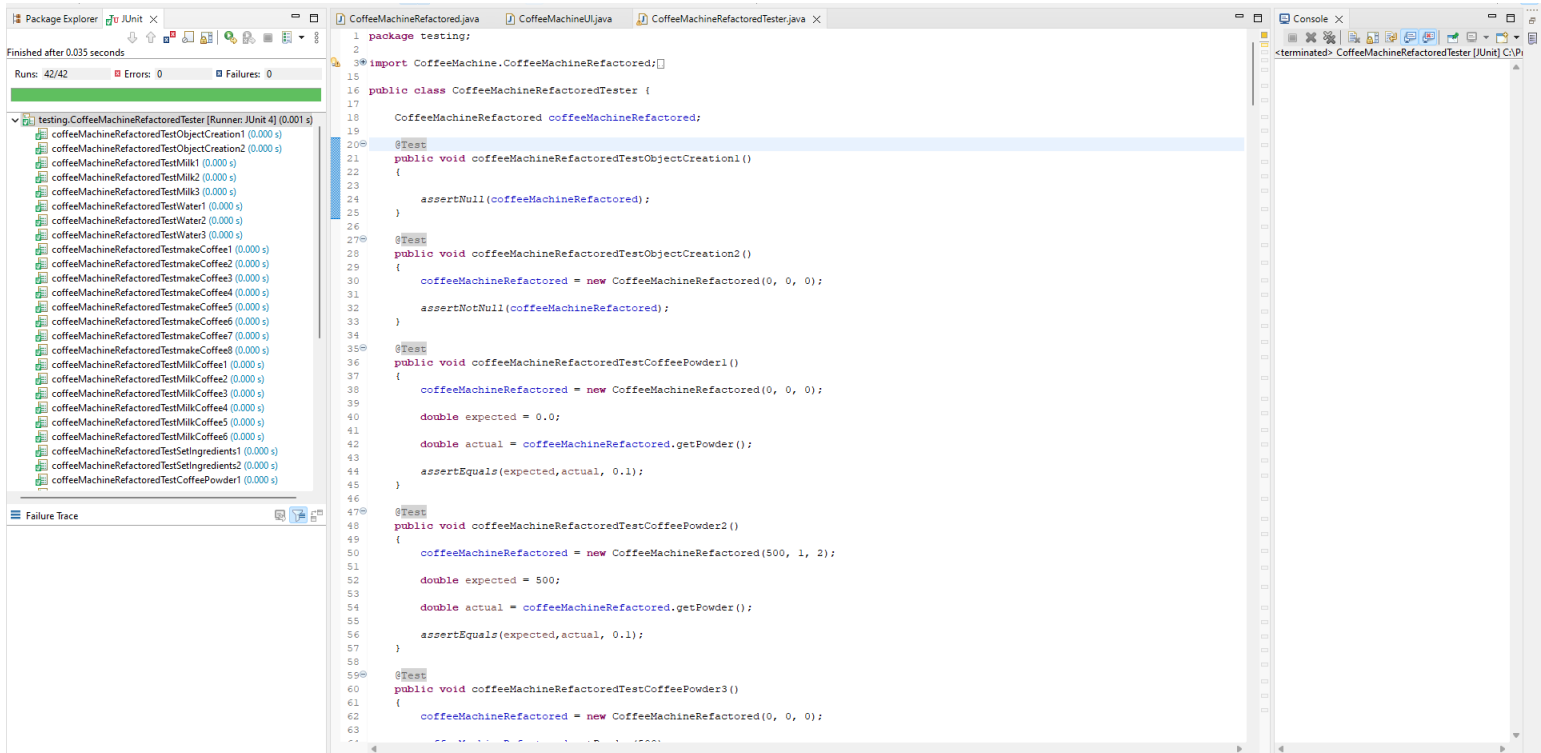


```
"CoffeeMachineRefactored.java" x "CoffeeMachineUI.java" x
52 System.out.println("\nCleaning Machine...");
53 CoffeeMachineRefactoredObject.CleanMachine();
54 System.out.println("Cleaning Completed.");
55 break;
56 case '4':
57 System.out.println("\n ----- ");
58 System.out.println("| Select Type: | \n ----- \n| 1: Black Coffee | \n| 2: Milk Coffee | \n| 0 to
59 System.out.println(" ----- \n");
60 char option = scan.next().charAt(0);
61 int result = CoffeeMachineRefactoredObject.makecoffee(option);
62 if(result == 1)
63 {
64 System.out.println("\n\nAvailable Coffee Power(Gram) "+String.format("%.1f",CoffeeMachineRefactoredObject.getPowder
65 System.out.println("Available Water(Liter) "+String.format("%.1f", CoffeeMachineRefactoredObject.getWater()));
66 }
67 else if(result == 2)
68 {
69 System.out.println("\n\nAvailable Coffee Power(Gram) "+String.format("%.1f",CoffeeMachineRefactoredObject.getPowder
70 System.out.println("Available Milk(Liter) "+String.format("%.1f", CoffeeMachineRefactoredObject.getMilk()));
71 System.out.println("Available Water(Liter) "+String.format("%.1f", CoffeeMachineRefactoredObject.getWater()));
72 }
73 else if((option == '1' || option == '2') && result == 0)
74 {
75 System.out.println("\n\nSome Items Are Not Available, Please Fill before Making Coffee.");
76 }
77 else if(option == '0' && result == 0)
78 {
79 //Discard
80 }
81 else
82 {
83 System.out.println("\n\nBad choice.");
84 }
85 break;
86 case '5':
87 System.out.println("\n\nWe Have Made "+CoffeeMachineRefactoredObject.getCoffee_Count()+" Coffees.");
88 break;
89 case '6':
90 System.out.println("\n\nExiting...\n");
91 t = false;
92 break;
93 }
94 }
95 }
96
97 public static void main(String[] args)
98 {
99 CoffeeMachineRefactored coffeeMachineObj = new CoffeeMachineRefactored(0,0,0);
100 CoffeeMachineUI coffeeMachineUIObj = new CoffeeMachineUI(coffeeMachineObj);
101
102 System.out.println("\n\nWants to Start Machine Y or N ?");
103
104 char d;
105 d = scan.next().charAt(0);
106
107 if(d == 'Y' || d == 'y')
108 {
109 CoffeeMachineUIObj.start(); //In order to call All Private Method Calling Public method
110 System.out.println("Shutting Down...\n");
111 }
112 else
113 System.out.println("Shutting Down...\n");
114 }
115 }
116
```



## Second: Use Junit to apply integration testing by

### 1- Make unit testing for CoffeeMachineRefactored class.



42 test cases all passed to test all functionality of the class.

The code and test cases of the CoffeeMachineRefactoredTester class

```
package testing;

import CoffeeMachine.CoffeeMachineRefactored;
import CoffeeMachine.CoffeeMachineUI;

import static org.junit.Assert.assertNotNull;
import static org.junit.Assert.assertNull;
import static org.junit.Assert.assertSame;
import static org.junit.Assert.assertTrue;
import static org.junit.Assert.assertEquals;
import static org.mockito.Mockito.doNothing;
import static org.mockito.Mockito.mock;

import org.junit.Test;

public class CoffeeMachineRefactoredTester {
```





```
CoffeeMachineRefactored coffeeMachineRefactored;

@Test
public void coffeeMachineRefactoredTestObjectCreation1()
{
    assertNull(coffeeMachineRefactored);
}

@Test
public void coffeeMachineRefactoredTestObjectCreation2()
{
    coffeeMachineRefactored = new CoffeeMachineRefactored(0, 0, 0);

    assertNotNull(coffeeMachineRefactored);
}

@Test
public void coffeeMachineRefactoredTestCoffeePowder1()
{
    coffeeMachineRefactored = new CoffeeMachineRefactored(0, 0, 0);

    double expected = 0.0;

    double actual = coffeeMachineRefactored.getPowder();

    assertEquals(expected, actual, 0.1);
}

@Test
public void coffeeMachineRefactoredTestCoffeePowder2()
{
    coffeeMachineRefactored = new CoffeeMachineRefactored(500, 1, 2);

    double expected = 500;

    double actual = coffeeMachineRefactored.getPowder();

    assertEquals(expected, actual, 0.1);
}

@Test
public void coffeeMachineRefactoredTestCoffeePowder3()
{

```



```
        coffeeMachineRefactored = new CoffeeMachineRefactored(0, 0, 0);

        coffeeMachineRefactored.setPowder(500);

        double expected = 500;

        double actual = coffeeMachineRefactored.getPowder();

        assertEquals(expected,actual, 0.1);
    }

    @Test
    public void coffeeMachineRefactoredTestMilk1()
    {
        coffeeMachineRefactored = new CoffeeMachineRefactored(0, 0, 0);

        double expected = 0.0;

        double actual = coffeeMachineRefactored.getMilk();

        assertEquals(expected,actual, 0.1);
    }

    @Test
    public void coffeeMachineRefactoredTestMilk2()
    {
        coffeeMachineRefactored = new CoffeeMachineRefactored(500, 1, 2);

        double expected = 1;

        double actual = coffeeMachineRefactored.getMilk();

        assertEquals(expected,actual,0.1);
    }

    @Test
    public void coffeeMachineRefactoredTestMilk3()
    {
        coffeeMachineRefactored = new CoffeeMachineRefactored(0, 0, 0);

        coffeeMachineRefactored.setMilk(1);

        double expected = 1;
```



```
        double actual = coffeeMachineRefactored.getMilk();

        assertEquals(expected,actual,0.1);
    }

    @Test
    public void coffeeMachineRefactoredTestWater1()
    {
        coffeeMachineRefactored = new CoffeeMachineRefactored(0, 0, 0);

        double expected = 0.0;

        double actual = coffeeMachineRefactored.getWater();

        assertEquals(expected,actual, 0.1);
    }

    @Test
    public void coffeeMachineRefactoredTestWater2()
    {
        coffeeMachineRefactored = new CoffeeMachineRefactored(500, 1, 2);

        double expected = 2;

        double actual = coffeeMachineRefactored.getWater();

        assertEquals(expected,actual, 0.1);
    }

    @Test
    public void coffeeMachineRefactoredTestWater3()
    {
        coffeeMachineRefactored = new CoffeeMachineRefactored(0, 0, 0);

        coffeeMachineRefactored.setWater(2);

        double expected = 2;

        double actual = coffeeMachineRefactored.getWater();

        assertEquals(expected,actual, 0.1);
    }
}
```



```
@Test
public void coffeeMachineRefactoredTestSetIngredients1()
{
    coffeeMachineRefactored = new CoffeeMachineRefactored(0, 0, 0);

    coffeeMachineRefactored.SetIngredient();

    boolean expected = true;

    boolean actual = coffeeMachineRefactored.getPowder() == 500.0 &&
coffeeMachineRefactored.getMilk() == 1.0
        && coffeeMachineRefactored.getWater() == 2.0;

    assertEquals(expected, actual);
}

@Test
public void coffeeMachineRefactoredTestSetIngredients2()
{
    coffeeMachineRefactored = new CoffeeMachineRefactored(50, 1, 1);

    coffeeMachineRefactored.SetIngredient();

    boolean expected = true;

    boolean actual = coffeeMachineRefactored.getPowder() == 500.0 &&
coffeeMachineRefactored.getMilk() == 1.0
        && coffeeMachineRefactored.getWater() == 2.0;

    assertEquals(expected, actual);
}

@Test
public void coffeeMachineRefactoredTestCleanMachine1()
{
    coffeeMachineRefactored = new CoffeeMachineRefactored(0, 0, 0);

    coffeeMachineRefactored.CleanMachine();

    boolean expected = true;

    boolean actual = coffeeMachineRefactored.getPowder() == 0.0 &&
coffeeMachineRefactored.getMilk() == 0.0
        && coffeeMachineRefactored.getWater() == 0.0;
```



```
        assertEquals(expected,actual);
    }

    @Test
    public void coffeeMachineRefactoredTestCleanMachine2()
    {
        coffeeMachineRefactored = new CoffeeMachineRefactored(500.0, 1.0, 2.0);

        coffeeMachineRefactored.CleanMachine();

        boolean expected = true;

        boolean actual = coffeeMachineRefactored.getPowder() == 0.0 &&
coffeeMachineRefactored.getMilk() == 0.0
                && coffeeMachineRefactored.getWater() == 0.0;

        assertEquals(expected,actual);
    }

    @Test
    public void coffeeMachineRefactoredTestBlackCoffee1()
    {
        coffeeMachineRefactored = new CoffeeMachineRefactored(0, 0, 0);

        boolean expected = false;

        boolean actual = coffeeMachineRefactored.BlackCoffee();

        assertEquals(expected,actual);
    }

    @Test
    public void coffeeMachineRefactoredTestBlackCoffee2()
    {
        coffeeMachineRefactored = new CoffeeMachineRefactored(0, 0, 0);

        coffeeMachineRefactored.SetIngredient();

        boolean expected = true;

        boolean actual = coffeeMachineRefactored.BlackCoffee();

        assertEquals(expected,actual);
    }
}
```



```
}

@Test
public void coffeeMachineRefactoredTestBlackCoffee3()
{
    coffeeMachineRefactored = new CoffeeMachineRefactored(500.0, 0, 0);

    boolean expected = false;

    boolean actual = coffeeMachineRefactored.BlackCoffee();

    assertEquals(expected,actual);
}

@Test
public void coffeeMachineRefactoredTestBlackCoffee4()
{
    coffeeMachineRefactored = new CoffeeMachineRefactored(0, 0, 2);

    boolean expected = false;

    boolean actual = coffeeMachineRefactored.BlackCoffee();

    assertEquals(expected,actual);
}

@Test
public void coffeeMachineRefactoredTestBlackCoffee5()
{
    coffeeMachineRefactored = new CoffeeMachineRefactored(0, 1, 0);

    boolean expected = false;

    boolean actual = coffeeMachineRefactored.BlackCoffee();

    assertEquals(expected,actual);
}

@Test
public void coffeeMachineRefactoredTestBlackCoffee6()
{
    coffeeMachineRefactored = new CoffeeMachineRefactored(0, 0, 0);
```



```
        coffeeMachineRefactored.setPowder(10);

        coffeeMachineRefactored.setWater(0.2);

        boolean expected = true;

        boolean actual = coffeeMachineRefactored.BlackCoffee();

        assertEquals(expected,actual);
    }

    @Test
    public void coffeeMachineRefactoredTestBlackCoffee7()
    {
        coffeeMachineRefactored = new CoffeeMachineRefactored(0, 0, 0);

        coffeeMachineRefactored.SetIngredient();

        coffeeMachineRefactored.BlackCoffee();

        boolean expected = true;

        boolean actual = coffeeMachineRefactored.getPowder() == 500.0 - 10.0 &&
            coffeeMachineRefactored.getWater() == 2.0 - 0.2 &&
coffeeMachineRefactored.getMilk() == 1.0;

        assertEquals(expected,actual);
    }

    @Test
    public void coffeeMachineRefactoredTestMilkCoffee1()
    {
        coffeeMachineRefactored = new CoffeeMachineRefactored(0, 0, 0);

        boolean expected = false;

        boolean actual = coffeeMachineRefactored.MilkCoffee();

        assertEquals(expected,actual);
    }
```



```
@Test
public void coffeeMachineRefactoredTestMilkCoffee2()
{
    coffeeMachineRefactored = new CoffeeMachineRefactored(0, 0, 0);

    coffeeMachineRefactored.SetIngredient();

    boolean expected = true;

    boolean actual = coffeeMachineRefactored.MilkCoffee();

    assertEquals(expected, actual);
}

@Test
public void coffeeMachineRefactoredTestMilkCoffee3()
{
    coffeeMachineRefactored = new CoffeeMachineRefactored(500.0, 0, 0);

    boolean expected = false;

    boolean actual = coffeeMachineRefactored.MilkCoffee();

    assertEquals(expected, actual);
}

@Test
public void coffeeMachineRefactoredTestMilkCoffee4()
{
    coffeeMachineRefactored = new CoffeeMachineRefactored(0, 0, 2);

    boolean expected = false;

    boolean actual = coffeeMachineRefactored.MilkCoffee();

    assertEquals(expected, actual);
}

@Test
public void coffeeMachineRefactoredTestMilkCoffee5()
{
    coffeeMachineRefactored = new CoffeeMachineRefactored(0, 1, 0);
```





```
        boolean expected = false;

        boolean actual = coffeeMachineRefactored.MilkCoffee();

        assertEquals(expected,actual);
    }

    @Test
    public void coffeeMachineRefactoredTestMilkCoffee6()
    {
        coffeeMachineRefactored = new CoffeeMachineRefactored(0, 0, 0);

        coffeeMachineRefactored.setPowder(10);

        coffeeMachineRefactored.setWater(0.2);

        coffeeMachineRefactored.setMilk(0.4);

        boolean expected = true;

        boolean actual = coffeeMachineRefactored.MilkCoffee();

        assertEquals(expected,actual);
    }

    @Test
    public void coffeeMachineRefactoredTestMilkoffee7()
    {
        coffeeMachineRefactored = new CoffeeMachineRefactored(0, 0, 0);

        coffeeMachineRefactored.SetIngredient();

        coffeeMachineRefactored.MilkCoffee();

        boolean expected = true;

        boolean actual = coffeeMachineRefactored.getPowder() == 500.0 - 10.0 &&
            coffeeMachineRefactored.getWater() == 2.0 - 0.2 &&
            coffeeMachineRefactored.getMilk() == 1.0 - 0.4;

        assertEquals(expected,actual);
    }
}
```



```
@Test
public void coffeeMachineRefactoredTestCoffeeCount1()
{
    coffeeMachineRefactored = new CoffeeMachineRefactored(0, 0, 0);

    int expected = 0;

    int actual = coffeeMachineRefactored.getCoffee_Count();

    assertEquals(expected,actual);
}

@Test
public void coffeeMachineRefactoredTestCoffeeCount2()
{
    coffeeMachineRefactored = new CoffeeMachineRefactored(0, 0, 0);

    coffeeMachineRefactored.BlackCoffee();

    int expected = 0;

    int actual = coffeeMachineRefactored.getCoffee_Count();

    assertEquals(expected,actual);
}

@Test
public void coffeeMachineRefactoredTestCoffeeCount3()
{
    coffeeMachineRefactored = new CoffeeMachineRefactored(0, 0, 0);

    coffeeMachineRefactored.MilkCoffee();

    int expected = 0;

    int actual = coffeeMachineRefactored.getCoffee_Count();

    assertEquals(expected,actual);
}

@Test
public void coffeeMachineRefactoredTestCoffeeCount4()
```



```
{
    coffeeMachineRefactored = new CoffeeMachineRefactored(0, 0, 0);

    coffeeMachineRefactored.SetIngredient();

    coffeeMachineRefactored.BlackCoffee();

    int expected = 1;

    int actual = coffeeMachineRefactored.getCoffee_Count();

    assertEquals(expected,actual);
}

@Test
public void coffeeMachineRefactoredTestCoffeeCount5()
{
    coffeeMachineRefactored = new CoffeeMachineRefactored(0, 0, 0);

    coffeeMachineRefactored.SetIngredient();

    coffeeMachineRefactored.MilkCoffee();

    int expected = 1;

    int actual = coffeeMachineRefactored.getCoffee_Count();

    assertEquals(expected,actual);
}

@Test
public void coffeeMachineRefactoredTestmakeCoffee1()
{
    coffeeMachineRefactored = new CoffeeMachineRefactored(0, 0, 0);

    int expected = 0;

    int actual = coffeeMachineRefactored.makecoffee('0');

    assertEquals(expected,actual);
}

@Test
```



```
public void coffeeMachineRefactoredTestmakeCoffee2()
{
    coffeeMachineRefactored = new CoffeeMachineRefactored(0, 0, 0);

    int expected = 5;

    int actual = coffeeMachineRefactored.makecoffee('5');

    assertEquals(expected,actual);
}

@Test
public void coffeeMachineRefactoredTestmakeCoffee3()
{
    coffeeMachineRefactored = new CoffeeMachineRefactored(0, 0, 0);

    int expected = 0;

    int actual = coffeeMachineRefactored.makecoffee('1');

    assertEquals(expected,actual);
}

@Test
public void coffeeMachineRefactoredTestmakeCoffee4()
{
    coffeeMachineRefactored = new CoffeeMachineRefactored(500.0, 1.0, 2.0);

    int expected = 1;

    int actual = coffeeMachineRefactored.makecoffee('1');

    assertEquals(expected,actual);
}

@Test
public void coffeeMachineRefactoredTestmakeCoffee5()
{
    coffeeMachineRefactored = new CoffeeMachineRefactored(0, 0, 0);

    coffeeMachineRefactored.SetIngredient();

    int expected = 1;
```



```
        int actual = coffeeMachineRefactored.makecoffee('1');

        assertEquals(expected,actual);
    }

    @Test
    public void coffeeMachineRefactoredTestmakeCoffee6()
    {
        coffeeMachineRefactored = new CoffeeMachineRefactored(0, 0, 0);

        int expected = 0;

        int actual = coffeeMachineRefactored.makecoffee('2');

        assertEquals(expected,actual);
    }

    @Test
    public void coffeeMachineRefactoredTestmakeCoffee7()
    {
        coffeeMachineRefactored = new CoffeeMachineRefactored(500.0, 1.0, 2.0);

        int expected = 2;

        int actual = coffeeMachineRefactored.makecoffee('2');

        assertEquals(expected,actual);
    }

    @Test
    public void coffeeMachineRefactoredTestmakeCoffee8()
    {
        coffeeMachineRefactored = new CoffeeMachineRefactored(0, 0, 0);

        coffeeMachineRefactored.SetIngredient();

        int expected = 2;

        int actual = coffeeMachineRefactored.makecoffee('2');

        assertEquals(expected,actual);
    }
}
```



## 2-Top-Down Approach

The screenshot shows the Eclipse IDE interface. On the left, the Package Explorer displays a list of test cases under the package `testing.CoffeeMachineUITopDownTester`. The test cases are:

- `coffeeMachineUITestBlackCoffee1` (0.545 s)
- `coffeeMachineUITestBlackCoffee2` (0.109 s)
- `coffeeMachineUITestBlackCoffee3` (0.016 s)
- `coffeeMachineUITestStatusOfIngredient1` (0.034 s)
- `coffeeMachineUITestStatusOfIngredient2` (0.016 s)
- `coffeeMachineUITestObjectCreation1` (0.009 s)
- `coffeeMachineUITestObjectCreation2` (0.009 s)
- `coffeeMachineUITestWhiteCoffee1` (0.014 s)
- `coffeeMachineUITestWhiteCoffee2` (0.012 s)
- `coffeeMachineUITestWhiteCoffee3` (0.011 s)
- `coffeeMachineUITestCoffeeBadChoice1` (0.011 s)
- `coffeeMachineUITestFillingIngredient1` (0.009 s)
- `coffeeMachineUITestCoffeeDiscard1` (0.010 s)
- `coffeeMachineUITestCoffeeCount1` (0.018 s)
- `coffeeMachineUITestCoffeeCount2` (0.010 s)
- `coffeeMachineUITestCoffeeCount3` (0.014 s)
- `coffeeMachineUITestCleanMachine1` (0.011 s)

The main editor shows the source code of `CoffeeMachineUITopDownTester.java`. The code is a Java class that implements a top-down testing approach for a coffee machine. It includes methods for starting the machine, filling it, cleaning it, and selecting coffee. The code is annotated with JUnit 4 annotations like `@RunWith(MockitoJUnitRunner.class)` and `@Before`.

17 test cases to make Top-Down integration testing by using stubs as shown in next snapshot.

```
132
133 @Before
134 public void InitBefore()
135 {
136     this.baos = new ByteArrayOutputStream();
137     PrintStream printStream = new PrintStream(baos);
138     System.setOut(printStream);
139
140     coffeeMachineRefactored = mock(CoffeeMachineRefactored.class);
141
142     coffeeMachineUI = mock(CoffeeMachineUI.class);
143
144     openMocks = MockitoAnnotations.openMocks(this);
145 }
146
147
148
149 @After
150 public void CloseAfter()
151 {
152     try
153     {
154         openMocks.close();
155         baos.close();
156     } catch (Exception e)
157     {
158         e.printStackTrace();
159     }
160 }
161
```



The code and test cases of *CoffeeMachineUITopDownTester* class.

```
package testing;

import CoffeeMachine.*;

import static org.junit.Assert.assertNotNull;
import static org.junit.Assert.assertNull;
import static org.junit.Assert.assertSame;
import static org.junit.Assert.assertTrue;
import static org.junit.Assert.assertEquals;
import static org.mockito.Mockito.doNothing;
import static org.mockito.Mockito.mock;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.PrintStream;

import org.junit.*;
import org.junit.runner.RunWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.Mockito;
import org.mockito.MockitoAnnotations;
import org.mockito.junit.MockitoJUnitRunner;

@RunWith(MockitoJUnitRunner.class)
public class CoffeeMachineUITopDownTester
{

    private ByteArrayInputStream bais;
    private ByteArrayOutputStream baos;
    private String userInput = "";

    String start_ask = "\nWants to Start Machine Y or N ?\r\n";
    String start =
        "\n ----- \r\n" +
        "|1:    Status of Ingredient    |\r\n" +
        "| ----- \r\n" +
        "|2:    Fill Ingredient          |\r\n" +
        "| ----- \r\n" +
        "|3:    Clean Machine           |\r\n" +
        "| ----- \r\n" +
        "|4:    Make Coffee             |\r\n" +
```



```
" ----- \n" +
"|5: How many Coffee We have made?|\n" +
" ----- \n" +
"|6:      Exit      |\r\n" +
" ----- \n\n\r\n";

String fillingMessage = "\nFilling...\r\n" + "Filling Completed.\r\n";

String cleaningMessage = "\nCleaning Machine...\r\n" + "Cleaning Completed.\r\n";

String selectCoffee = "\n ----- \r\n" +
"|  Select Type:  |\n" +
" ----- \n" +
"| 1:  Black Coffee  |\n" +
"| 2:  Milk Coffee   |\n" +
"| 0   to Discard   |\r\n" +
" ----- \n\r\n";

String itemNotAvailable = "\nSome Items Are Not Available, Please Fill before Making Coffee.\r\n";

String badChoice = "\nBad choice.\r\n";

String exit_main = "\nExiting...\n\r\n" + "Shutting Down...\n\r\n";

String exit_start = "\nExiting...\n\r\n";

@Mock
private CoffeeMachineRefactored coffeeMachineRefactored;

@InjectMocks
private CoffeeMachineUI coffeeMachineUI;

AutoCloseable openMocks;

public String GetStartStatus()
```





```
{
    String start_status = " -----\r\n" +
        "|           Coffee Machine By Manikant           |\r\n" +
        " -----\r\n\r\n" +
        "Current Status: \r\n" +
        "Available Coffee Power(Gram) " + String.format("%.1f", coffeeMachineRefactored.getPowder()) +
        "\r\n" +
        "Available Milk(Liter) " + String.format("%.1f", coffeeMachineRefactored.getMilk()) + "\r\n" +
        "Available Water(Liter) " + String.format("%.1f", coffeeMachineRefactored.getWater()) + "\r\n";

    return start_status;
}

public String GetStatus()
{
    String status = "----- Status ----- \r\n" +
        "Available Coffee Power(Gram) " + String.format("%.1f", coffeeMachineRefactored.getPowder()) +
        "\r\n" +
        "Available Milk(Liter) " + String.format("%.1f", coffeeMachineRefactored.getMilk()) + "\r\n" +
        "Available Water(Liter) " + String.format("%.1f", coffeeMachineRefactored.getWater()) + "\r\n" +
        "----- \r\n";

    return status;
}

public String BlackCoffeeMade()
{
    String statusMessage = "\nAvailable Coffee Power(Gram) " + String.format("%.1f",
        coffeeMachineRefactored.getPowder()) + "\r\n" +
        "Available Water(Liter) " + String.format("%.1f", coffeeMachineRefactored.getWater()) + "\r\n";

    return statusMessage;
}

public String WhiteCoffeeMade()
{
    String statusMessage = "\nAvailable Coffee Power(Gram) " + String.format("%.1f",
        coffeeMachineRefactored.getPowder()) + "\r\n" +
        "Available Milk(Liter) " + String.format("%.1f", coffeeMachineRefactored.getMilk()) + "\r\n" +
        "Available Water(Liter) " + String.format("%.1f", coffeeMachineRefactored.getWater()) + "\r\n";
}
```



```
        return statusMessage;
    }

    public String GetCoffeeCount()
    {
        String message = "\nWe Have Made " + coffeeMachineRefactored.getCoffee_Count() + " Coffees.\r\n";

        return message;
    }

    @Before
    public void InitBefore()
    {
        this.baos = new ByteArrayOutputStream();
        PrintStream printStream = new PrintStream(baos);
        System.setOut(printStream);

        coffeeMachineRefactored = mock(CoffeeMachineRefactored.class);

        coffeeMachineUI = mock(CoffeeMachineUI.class);

        openMocks = MockitoAnnotations.openMocks(this);
    }

    @After
    public void CloseAfter()
    {
        try
        {
            openMocks.close();
            baos.close();
        } catch (Exception e)
        {
            e.printStackTrace();
        }
    }

    @Test
    public void coffeeMachineUITestObjectCreation1()
    {
```



```
        assertNotNull(coffeeMachineUI);
    }

    @Test
    public void coffeeMachineUITestObjectCreation2()
    {
        coffeeMachineUI = new CoffeeMachineUI(coffeeMachineRefactored);

        assertNotNull(coffeeMachineUI);
    }

    @Test
    public void coffeeMachineUITestStatusOfIngredient1()
    {
        Mockito.when(coffeeMachineRefactored.getPowder()).thenReturn(0.0);
        Mockito.when(coffeeMachineRefactored.getMilk()).thenReturn(0.0);
        Mockito.when(coffeeMachineRefactored.getWater()).thenReturn(0.0);

        String expected = GetStartStatus() + start + GetStatus() + start + exit_start;

        userInput = "1\r6";

        bais = new ByteArrayInputStream(userInput.getBytes());

        System.setIn(bais);

        coffeeMachineUI = new CoffeeMachineUI(coffeeMachineRefactored);

        coffeeMachineUI.start();

        String actual = baos.toString();

        assertEquals(expected, actual);
    }

    @Test
    public void coffeeMachineUITestStatusOfIngredient2()
    {
        Mockito.when(coffeeMachineRefactored.getPowder()).thenReturn(50.0);
        Mockito.when(coffeeMachineRefactored.getMilk()).thenReturn(2.0);
        Mockito.when(coffeeMachineRefactored.getWater()).thenReturn(3.0);

        String expected = GetStartStatus() + start + GetStatus() + start + exit_start;
```



```
        userInput = "1\r6";

        bais = new ByteArrayInputStream(userInput.getBytes());

        System.setIn(bais);

        coffeeMachineUI = new CoffeeMachineUI(coffeeMachineRefactored);

        coffeeMachineUI.start();

        String actual = baos.toString();

        assertEquals(expected, actual);
    }

    @Test
    public void coffeeMachineUITestFillIngredient1()
    {

        String expected = GetStartStatus() + start + fillingMessage + start + exit_start;

        userInput = "2\r6";

        bais = new ByteArrayInputStream(userInput.getBytes());

        System.setIn(bais);

        coffeeMachineUI = new CoffeeMachineUI(coffeeMachineRefactored);

        coffeeMachineUI.start();

        String actual = baos.toString();

        assertEquals(expected, actual);
    }

    @Test
    public void coffeeMachineUITestCleanMachine1()
    {

        String expected = GetStartStatus() + start + cleaningMessage + start + exit_start;

        userInput = "3\r6";
```



```
bais = new ByteArrayInputStream(userInput.getBytes());

System.setIn(bais);

coffeeMachineUI = new CoffeeMachineUI(coffeeMachineRefactored);

coffeeMachineUI.start();

String actual = baos.toString();

assertEquals(expected, actual);
}

@Test
public void coffeeMachineUITestCoffeeDiscard1()
{

    String expected = GetStartStatus() + start + selectCoffee + start + exit_start;

    userInput = "4\r0\r6";

    bais = new ByteArrayInputStream(userInput.getBytes());

    System.setIn(bais);

    coffeeMachineUI = new CoffeeMachineUI(coffeeMachineRefactored);

    coffeeMachineUI.start();

    String actual = baos.toString();

    assertEquals(expected, actual);
}

@Test
public void coffeeMachineUITestCoffeeBadChoice1()
{

    String expected = GetStartStatus() + start + selectCoffee + badChoice + start + exit_start;

    userInput = "4\r5\r6";

    bais = new ByteArrayInputStream(userInput.getBytes());
```



```
        System.setIn(bais);

        coffeeMachineUI = new CoffeeMachineUI(coffeeMachineRefactored);

        coffeeMachineUI.start();

        String actual = baos.toString();

        assertEquals(expected, actual);
    }

    @Test
    public void coffeeMachineUITestBlackCoffee1()
    {

        String expected = GetStartStatus() + start + selectCoffee + itemNotAvailable + start + exit_start;

        userInput = "4\r1\r6";

        bais = new ByteArrayInputStream(userInput.getBytes());

        System.setIn(bais);

        coffeeMachineUI = new CoffeeMachineUI(coffeeMachineRefactored);

        coffeeMachineUI.start();

        String actual = baos.toString();

        assertEquals(expected, actual);
    }

    @Test
    public void coffeeMachineUITestBlackCoffee2()
    {

        Mockito.when(coffeeMachineRefactored.makecoffee(Mockito.anyChar())).thenReturn(1);

        String expected = GetStartStatus() + start + fillingMessage + start + selectCoffee + BlackCoffeeMade() +
start + exit_start;

        userInput = "2\r4\r1\r6";

        bais = new ByteArrayInputStream(userInput.getBytes());
```



```
        System.setIn(bais);

        coffeeMachineUI = new CoffeeMachineUI(coffeeMachineRefactored);

        coffeeMachineUI.start();

        String actual = baos.toString();

        assertEquals(expected, actual);
    }

    @Test
    public void coffeeMachineUITestBlackCoffee3()
    {
        Mockito.when(coffeeMachineRefactored.makecoffee(Mockito.anyChar())).thenReturn(0);

        String expected = GetStartStatus() + start + fillingMessage + start + selectCoffee + itemNotAvailable +
start + exit_start;

        userInput = "2\r4\r1\r6";

        bais = new ByteArrayInputStream(userInput.getBytes());

        System.setIn(bais);

        coffeeMachineUI = new CoffeeMachineUI(coffeeMachineRefactored);

        coffeeMachineUI.start();

        String actual = baos.toString();

        assertEquals(expected, actual);
    }

    @Test
    public void coffeeMachineUITestWhiteCoffee1()
    {
        String expected = GetStartStatus() + start + selectCoffee + itemNotAvailable + start + exit_start;

        userInput = "4\r2\r6";

        bais = new ByteArrayInputStream(userInput.getBytes());
```



```
        System.setIn(bais);

        coffeeMachineUI = new CoffeeMachineUI(coffeeMachineRefactored);

        coffeeMachineUI.start();

        String actual = baos.toString();

        assertEquals(expected, actual);
    }

    @Test
    public void coffeeMachineUITestWhiteCoffee2()
    {
        Mockito.when(coffeeMachineRefactored.makecoffee(Mockito.anyChar())).thenReturn(2);

        String expected = GetStartStatus() + start + fillingMessage + start + selectCoffee + WhiteCoffeeMade() +
start + exit_start;

        userInput = "2\r4\r2\r6";

        bais = new ByteArrayInputStream(userInput.getBytes());

        System.setIn(bais);

        coffeeMachineUI = new CoffeeMachineUI(coffeeMachineRefactored);

        coffeeMachineUI.start();

        String actual = baos.toString();

        assertEquals(expected, actual);
    }

    @Test
    public void coffeeMachineUITestWhiteCoffee3()
    {
        Mockito.when(coffeeMachineRefactored.makecoffee(Mockito.anyChar())).thenReturn(0);

        String expected = GetStartStatus() + start + fillingMessage + start + selectCoffee + itemNotAvailable +
start + exit_start;

        userInput = "2\r4\r2\r6";
```





```
bais = new ByteArrayInputStream(userInput.getBytes());

System.setIn(bais);

coffeeMachineUI = new CoffeeMachineUI(coffeeMachineRefactored);

coffeeMachineUI.start();

String actual = baos.toString();

assertEquals(expected, actual);
}

@Test
public void coffeeMachineUITestCoffeeCount1()
{
    Mockito.when(coffeeMachineRefactored.getCoffee_Count()).thenReturn(0);

    String expected = GetStartStatus() + start + GetCoffeeCount() + start + exit_start;

    userInput = "5\r6\r";

    bais = new ByteArrayInputStream(userInput.getBytes());

    System.setIn(bais);

    coffeeMachineUI = new CoffeeMachineUI(coffeeMachineRefactored);

    coffeeMachineUI.start();

    String actual = baos.toString();

    assertEquals(expected, actual);
}

@Test
public void coffeeMachineUITestCoffeeCount2()
{
    Mockito.when(coffeeMachineRefactored.getCoffee_Count()).thenReturn(1);

    String expected = GetStartStatus() + start + GetCoffeeCount() + start + exit_start;

    userInput = "5\r6\r";
```



```
bais = new ByteArrayInputStream(userInput.getBytes());

System.setIn(bais);

coffeeMachineUI = new CoffeeMachineUI(coffeeMachineRefactored);

coffeeMachineUI.start();

String actual = baos.toString();

assertEquals(expected, actual);
}

@Test
public void coffeeMachineUITestCoffeeCount3()
{
    Mockito.when(coffeeMachineRefactored.getCoffee_Count()).thenReturn(5);

    String expected = GetStartStatus() + start + GetCoffeeCount() + start + exit_start;

    userInput = "5\r6\r";

    bais = new ByteArrayInputStream(userInput.getBytes());

    System.setIn(bais);

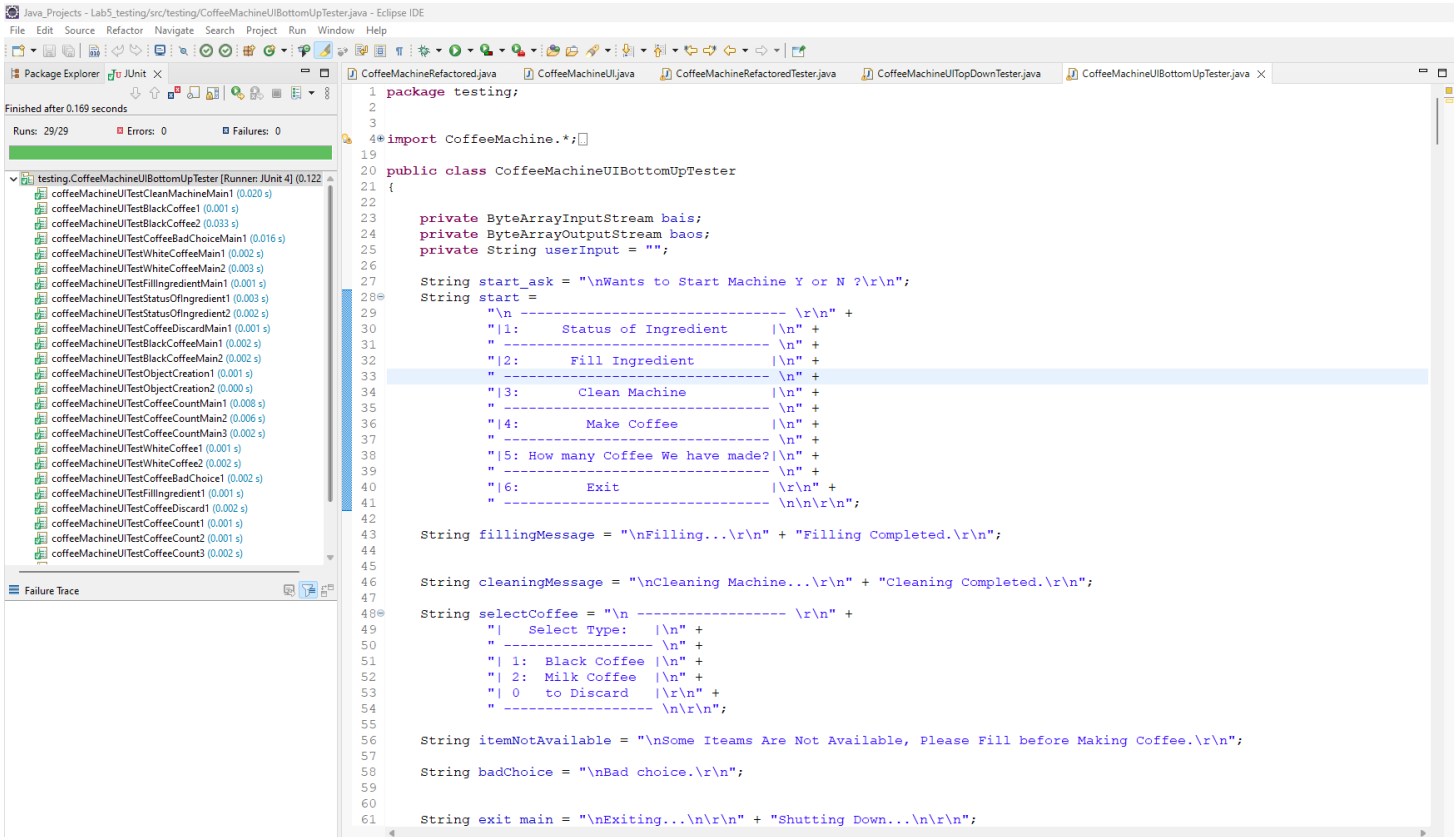
    coffeeMachineUI = new CoffeeMachineUI(coffeeMachineRefactored);

    coffeeMachineUI.start();

    String actual = baos.toString();

    assertEquals(expected, actual);
}
}
```

### 3- Bottom-Up Approach



The screenshot shows the Eclipse IDE with the following components:

- Package Explorer:** Lists test cases under the package `testing.CoffeeMachineUIBottomUpTester`. All 29 test cases passed, with a total duration of 0.169 seconds.
- Source Editor:** Displays the code for `CoffeeMachineUIBottomUpTester.java`. The code includes imports, class definition, and various test methods for different coffee machine operations.
- Failure Trace:** Shows no failures.

29 test cases all passed to make Bottom-Up integration testing by creating objects as shown.

```

122
123 @Before
124 public void InitBefore()
125 {
126     this.baos = new ByteArrayOutputStream();
127     PrintStream printStream = new PrintStream(baos);
128     System.setOut(printStream);
129
130     coffeeMachineRefactored = new CoffeeMachineRefactored(0, 0, 0);
131
132     coffeeMachineUI = new CoffeeMachineUI(coffeeMachineRefactored);
133 }
134
135
136
137 @After
138 public void CloseAfter()
139 {
140
141     try
142     {
143         baos.close();
144     } catch (Exception e)
145     {
146         e.printStackTrace();
147     }
148 }

```



The code and test cases of *CoffeeMachineUIBottomUpTester* class.

```
package testing;

import CoffeeMachine.*;

import static org.junit.Assert.assertNotNull;
import static org.junit.Assert.assertNull;
import static org.junit.Assert.assertSame;
import static org.junit.Assert.assertTrue;
import static org.junit.Assert.assertEquals;
import static org.mockito.Mockito.doNothing;
import static org.mockito.Mockito.mock;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.PrintStream;

import org.junit.*;

public class CoffeeMachineUIBottomUpTester
{

    private ByteArrayInputStream bais;
    private ByteArrayOutputStream baos;
    private String userInput = "";

    String start_ask = "\nWants to Start Machine Y or N ?\r\n";
    String start =
        "\n ----- \r\n" +
        "|1:    Status of Ingredient    |\r\n" +
        " ----- \r\n" +
        "|2:    Fill Ingredient         |\r\n" +
        " ----- \r\n" +
        "|3:    Clean Machine          |\r\n" +
        " ----- \r\n" +
        "|4:    Make Coffee            |\r\n" +
        " ----- \r\n" +
        "|5: How many Coffee We have made?|\r\n" +
        " ----- \r\n" +
        "|6:    Exit                   |\r\n" +
        " ----- \r\n\r\n";

    String fillingMessage = "\nFilling...\r\n" + "Filling Completed.\r\n";
```



```
String cleaningMessage = "\nCleaning Machine...\r\n" + "Cleaning Completed.\r\n";

String selectCoffee = "\n ----- \r\n" +
    "|   Select Type:   |\r\n" +
    "| ----- \r\n" +
    "| 1:  Black Coffee |\r\n" +
    "| 2:  Milk Coffee  |\r\n" +
    "| 0   to Discard   |\r\n" +
    "| ----- \r\n\r\n";

String itemNotAvailable = "\nSome Items Are Not Available, Please Fill before Making Coffee.\r\n";

String badChoice = "\nBad choice.\r\n";

String exit_main = "\nExiting...\r\n\r\n" + "Shutting Down...\r\n\r\n";

String exit_start = "\nExiting...\r\n\r\n";

private CoffeeMachineRefactored coffeeMachineRefactored;

private CoffeeMachineUI coffeeMachineUI;

public String GetStartStatus()
{
    String start_status = " -----
\r\n" +
        "|                               |\r\n" +
        "| ----- \r\n" +
        "|Current Status: \r\n" +
        "|Available Coffee Power(Gram) " + String.format("%.1f",
coffeeMachineRefactored.getPowder()) + "\r\n" +
        "|Available Milk(Liter) " + String.format("%.1f",
coffeeMachineRefactored.getMilk()) + "\r\n" +
        "|Available Water(Liter) " + String.format("%.1f",
coffeeMachineRefactored.getWater()) + "\r\n";

    return start_status;
}
```



```
public String GetStatus()
{
    String status = "----- Status -----\r\n" +
        "Available Coffee Power(Gram) " + String.format("%.1f",
coffeeMachineRefactored.getPowder()) + "\r\n" +
        "Available Milk(Liter) " + String.format("%.1f",
coffeeMachineRefactored.getMilk()) + "\r\n" +
        "Available Water(Liter) " + String.format("%.1f",
coffeeMachineRefactored.getWater()) + "\r\n" +
        "-----\r\n";

    return status;
}

public String BlackCoffeeMade()
{
    String statusMessage = "\nAvailable Coffee Power(Gram) " + String.format("%.1f",
coffeeMachineRefactored.getPowder()) + "\r\n" +
        "Available Water(Liter) " + String.format("%.1f",
coffeeMachineRefactored.getWater()) + "\r\n";

    return statusMessage;
}

public String WhiteCoffeeMade()
{
    String statusMessage = "\nAvailable Coffee Power(Gram) " + String.format("%.1f",
coffeeMachineRefactored.getPowder()) + "\r\n" +
        "Available Milk(Liter) " + String.format("%.1f",
coffeeMachineRefactored.getMilk()) + "\r\n" +
        "Available Water(Liter) " + String.format("%.1f",
coffeeMachineRefactored.getWater()) + "\r\n";

    return statusMessage;
}

public String GetCoffeeCount()
{

```



```
        String message = "\nWe Have Made " + coffeeMachineRefactored.getCoffee_Count() + "
        Coffees.\r\n";

        return message;
    }

    @Before
    public void InitBefore()
    {
        this.baos = new ByteArrayOutputStream();
        PrintStream printStream = new PrintStream(baos);
        System.setOut(printStream);

        coffeeMachineRefactored = new CoffeeMachineRefactored(0, 0, 0);

        coffeeMachineUI = new CoffeeMachineUI(coffeeMachineRefactored);
    }

    @After
    public void CloseAfter()
    {
        try
        {
            baos.close();
        } catch (Exception e)
        {
            e.printStackTrace();
        }
    }

    @Test
    public void coffeeMachineUITestObjectCreation1()
    {
        assertNotNull(coffeeMachineUI);
    }

    @Test
    public void coffeeMachineUITestObjectCreation2()
    {
        coffeeMachineUI = new CoffeeMachineUI(coffeeMachineRefactored);
    }
}
```



```
        assertNotNull(coffeeMachineUI);
    }

    @Test
    public void coffeeMachineUITestStatusOfIngredient1()
    {

        String expected = GetStartStatus() + start + GetStatus() + start + exit_start;

        userInput = "1\r6";

        bais = new ByteArrayInputStream(userInput.getBytes());

        System.setIn(bais);

        coffeeMachineUI = new CoffeeMachineUI(coffeeMachineRefactored);

        coffeeMachineUI.start();

        String actual = baos.toString();

        assertEquals(expected, actual);
    }

    @Test
    public void coffeeMachineUITestStatusOfIngredient2()
    {

        String expected = GetStartStatus() + start + GetStatus() + start + exit_start;

        userInput = "1\r6";

        bais = new ByteArrayInputStream(userInput.getBytes());

        System.setIn(bais);

        coffeeMachineUI = new CoffeeMachineUI(coffeeMachineRefactored);

        coffeeMachineUI.start();

        String actual = baos.toString();

        assertEquals(expected, actual);
    }
}
```





```
@Test
public void coffeeMachineUITestFillIngredient1()
{

    String expected = GetStartStatus() + start + fillingMessage + start + exit_start;

    userInput = "2\r6";

    bais = new ByteArrayInputStream(userInput.getBytes());

    System.setIn(bais);

    coffeeMachineUI = new CoffeeMachineUI(coffeeMachineRefactored);

    coffeeMachineUI.start();

    String actual = baos.toString();

    assertEquals(expected, actual);
}

@Test
public void coffeeMachineUITestCleanMachine1()
{

    String expected = GetStartStatus() + start + cleaningMessage + start + exit_start;

    userInput = "3\r6";

    bais = new ByteArrayInputStream(userInput.getBytes());

    System.setIn(bais);

    coffeeMachineUI = new CoffeeMachineUI(coffeeMachineRefactored);

    coffeeMachineUI.start();

    String actual = baos.toString();

    assertEquals(expected, actual);
}

@Test
```



```
public void coffeeMachineUITestCoffeeDiscard1()
{
    String expected = GetStartStatus() + start + selectCoffee + start + exit_start;

    userInput = "4\r0\r6";

    bais = new ByteArrayInputStream(userInput.getBytes());

    System.setIn(bais);

    coffeeMachineUI = new CoffeeMachineUI(coffeeMachineRefactored);

    coffeeMachineUI.start();

    String actual = baos.toString();

    assertEquals(expected, actual);
}

@Test
public void coffeeMachineUITestCoffeeBadChoice1()
{
    String expected = GetStartStatus() + start + selectCoffee + badChoice + start +
exit_start;

    userInput = "4\r5\r6";

    bais = new ByteArrayInputStream(userInput.getBytes());

    System.setIn(bais);

    coffeeMachineUI = new CoffeeMachineUI(coffeeMachineRefactored);

    coffeeMachineUI.start();

    String actual = baos.toString();

    assertEquals(expected, actual);
}

@Test
public void coffeeMachineUITestBlackCoffee1()
```



```
{

    String expected = GetStartStatus() + start + selectCoffee + itemNotAvailable + start +
exit_start;

    userInput = "4\r1\r6";

    bais = new ByteArrayInputStream(userInput.getBytes());

    System.setIn(bais);

    coffeeMachineUI = new CoffeeMachineUI(coffeeMachineRefactored);

    coffeeMachineUI.start();

    String actual = baos.toString();

    assertEquals(expected, actual);
}

@Test
public void coffeeMachineUITestBlackCoffee2()
{
    String expected = GetStartStatus() + start + fillingMessage + start + selectCoffee;

    coffeeMachineRefactored.setPowder(490.0);

    coffeeMachineRefactored.setWater(1.8);

    expected += BlackCoffeeMade() + start + exit_start;

    userInput = "2\r4\r1\r6";

    bais = new ByteArrayInputStream(userInput.getBytes());

    System.setIn(bais);

    coffeeMachineRefactored = new CoffeeMachineRefactored(0, 0, 0);

    coffeeMachineUI = new CoffeeMachineUI(coffeeMachineRefactored);

    coffeeMachineUI.start();

    String actual = baos.toString();
```



```
        assertEquals(expected, actual);
    }

    @Test
    public void coffeeMachineUITestWhiteCoffee1()
    {
        String expected = GetStartStatus() + start + selectCoffee + itemNotAvailable + start +
exit_start;

        userInput = "4\r2\r6";

        bais = new ByteArrayInputStream(userInput.getBytes());

        System.setIn(bais);

        coffeeMachineUI = new CoffeeMachineUI(coffeeMachineRefactored);

        coffeeMachineUI.start();

        String actual = baos.toString();

        assertEquals(expected, actual);
    }

    @Test
    public void coffeeMachineUITestWhiteCoffee2()
    {
        String expected = GetStartStatus() + start + fillingMessage + start + selectCoffee;

        coffeeMachineRefactored.setPowder(490.0);

        coffeeMachineRefactored.setMilk(0.6);

        coffeeMachineRefactored.setWater(1.8);

        expected += WhiteCoffeeMade() + start + exit_start;

        userInput = "2\r4\r2\r6";

        bais = new ByteArrayInputStream(userInput.getBytes());
```



```
        System.setIn(bais);

        coffeeMachineRefactored = new CoffeeMachineRefactored(0, 0, 0);

        coffeeMachineUI = new CoffeeMachineUI(coffeeMachineRefactored);

        coffeeMachineUI.start();

        String actual = baos.toString();

        assertEquals(expected, actual);;
    }

    @Test
    public void coffeeMachineUITestCoffeeCount1()
    {

        String expected = GetStartStatus() + start + GetCoffeeCount() + start + exit_start;

        userInput = "5\r6\r";

        bais = new ByteArrayInputStream(userInput.getBytes());

        System.setIn(bais);

        coffeeMachineUI = new CoffeeMachineUI(coffeeMachineRefactored);

        coffeeMachineUI.start();

        String actual = baos.toString();

        assertEquals(expected, actual);;
    }

    @Test
    public void coffeeMachineUITestCoffeeCount2()
    {

        String expected = GetStartStatus() + start + GetCoffeeCount() + start + exit_start;

        userInput = "5\r6\r";
```



```
bais = new ByteArrayInputStream(userInput.getBytes());

System.setIn(bais);

coffeeMachineUI = new CoffeeMachineUI(coffeeMachineRefactored);

coffeeMachineUI.start();

String actual = baos.toString();

assertEquals(expected, actual);
}

@Test
public void coffeeMachineUITestCoffeeCount3()
{

    String expected = GetStartStatus() + start + GetCoffeeCount() + start + exit_start;

    userInput = "5\r6\r";

    bais = new ByteArrayInputStream(userInput.getBytes());

    System.setIn(bais);

    coffeeMachineUI = new CoffeeMachineUI(coffeeMachineRefactored);

    coffeeMachineUI.start();

    String actual = baos.toString();

    assertEquals(expected, actual);
}

@Test
public void coffeeMachineUITestStatusOfIngredientMain1()
{

    String expected = start_ask + GetStartStatus() + start + GetStatus() + start + exit_main;

    userInput = "Y\r1\r6";

    bais = new ByteArrayInputStream(userInput.getBytes());
```



```
        System.setIn(bais);

        CoffeeMachineUI.main(null);

        String actual = baos.toString();

        assertEquals(expected, actual);
    }

    @Test
    public void coffeeMachineUITestStatusOfIngredientMain2()
    {

        String expected = start_ask + GetStartStatus() + start + GetStatus() + start + exit_main;

        userInput = "Y\r1\r6";

        bais = new ByteArrayInputStream(userInput.getBytes());

        System.setIn(bais);

        CoffeeMachineUI.main(null);

        String actual = baos.toString();

        assertEquals(expected, actual);
    }

    @Test
    public void coffeeMachineUITestFillIngredientMain1()
    {

        String expected = start_ask + GetStartStatus() + start + fillingMessage + start +
exit_main;

        userInput = "Y\r2\r6";

        bais = new ByteArrayInputStream(userInput.getBytes());

        System.setIn(bais);

        CoffeeMachineUI.main(null);

        String actual = baos.toString();
```



```
        assertEquals(expected, actual);;
    }

    @Test
    public void coffeeMachineUITestCleanMachineMain1()
    {

        String expected = start_ask + GetStartStatus() + start + cleaningMessage + start +
exit_main;

        userInput = "Y\r3\r6";

        bais = new ByteArrayInputStream(userInput.getBytes());

        System.setIn(bais);

        CoffeeMachineUI.main(null);

        String actual = baos.toString();

        assertEquals(expected, actual);;
    }

    @Test
    public void coffeeMachineUITestCoffeeDiscardMain1()
    {

        String expected = start_ask + GetStartStatus() + start + selectCoffee + start + exit_main;

        userInput = "Y\r4\r0\r6";

        bais = new ByteArrayInputStream(userInput.getBytes());

        System.setIn(bais);

        CoffeeMachineUI.main(null);

        String actual = baos.toString();

        assertEquals(expected, actual);;
    }

    @Test
```





```
public void coffeeMachineUITestCoffeeBadChoiceMain1()
{
    String expected = start_ask + GetStartStatus() + start + selectCoffee + badChoice + start
+ exit_main;

    userInput = "Y\r4\r5\r6";

    bais = new ByteArrayInputStream(userInput.getBytes());

    System.setIn(bais);

    CoffeeMachineUI.main(null);

    String actual = baos.toString();

    assertEquals(expected, actual);
}

@Test
public void coffeeMachineUITestBlackCoffeeMain1()
{
    String expected = start_ask + GetStartStatus() + start + selectCoffee + itemNotAvailable +
start + exit_main;

    userInput = "Y\r4\r1\r6";

    bais = new ByteArrayInputStream(userInput.getBytes());

    System.setIn(bais);

    CoffeeMachineUI.main(null);

    String actual = baos.toString();

    assertEquals(expected, actual);
}

@Test
public void coffeeMachineUITestBlackCoffeeMain2()
{
    String expected = start_ask + GetStartStatus() + start + fillingMessage + start +
selectCoffee;
```



```
        coffeeMachineRefactored.setPowder(490.0);

        coffeeMachineRefactored.setWater(1.8);

        expected += BlackCoffeeMade() + start + exit_main;

        userInput = "Y\r2\r4\r1\r6";

        bais = new ByteArrayInputStream(userInput.getBytes());

        System.setIn(bais);

        CoffeeMachineUI.main(null);

        String actual = baos.toString();

        assertEquals(expected, actual);
    }

    @Test
    public void coffeeMachineUITestWhiteCoffeeMain1()
    {

        String expected = start_ask + GetStartStatus() + start + selectCoffee + itemNotAvailable +
start + exit_main;

        userInput = "Y\r4\r2\r6";

        bais = new ByteArrayInputStream(userInput.getBytes());

        System.setIn(bais);

        CoffeeMachineUI.main(null);

        String actual = baos.toString();

        assertEquals(expected, actual);
    }

    @Test
    public void coffeeMachineUITestWhiteCoffeeMain2()
```



```
{
    String expected = start_ask + GetStartStatus() + start + fillingMessage + start +
selectCoffee;

    coffeeMachineRefactored.setPowder(490.0);

    coffeeMachineRefactored.setMilk(0.6);

    coffeeMachineRefactored.setWater(1.8);

    expected += WhiteCoffeeMade() + start + exit_main;

    userInput = "Y\r2\r4\r2\r6";

    bais = new ByteArrayInputStream(userInput.getBytes());

    System.setIn(bais);

    CoffeeMachineUI.main(null);

    String actual = baos.toString();

    assertEquals(expected, actual);
}

@Test
public void coffeeMachineUITestCoffeeCountMain1()
{
    String expected = start_ask + GetStartStatus() + start + GetCoffeeCount() + start +
exit_main;

    userInput = "Y\r5\r6\r";

    bais = new ByteArrayInputStream(userInput.getBytes());

    System.setIn(bais);

    CoffeeMachineUI.main(null);

    String actual = baos.toString();

    assertEquals(expected, actual);
}
```



```
}

@Test
public void coffeeMachineUITestCoffeeCountMain2()
{
    String expected = start_ask + GetStartStatus() + start + GetCoffeeCount() + start +
exit_main;

    userInput = "Y\r5\r6\r";

    bais = new ByteArrayInputStream(userInput.getBytes());

    System.setIn(bais);

    CoffeeMachineUI.main(null);

    String actual = baos.toString();

    assertEquals(expected, actual);
}

@Test
public void coffeeMachineUITestCoffeeCountMain3()
{
    String expected = start_ask + GetStartStatus() + start + GetCoffeeCount() + start +
exit_main;

    userInput = "Y\r5\r6\r";

    bais = new ByteArrayInputStream(userInput.getBytes());

    System.setIn(bais);

    CoffeeMachineUI.main(null);

    String actual = baos.toString();

    assertEquals(expected, actual);
}

@Test
public void coffeeMachineUITestEndMain1()
```



```
{  
  
    String expected = start_ask + "Shutting Down...\n\r\n";  
  
    userInput = "N\r";  
  
    bais = new ByteArrayInputStream(userInput.getBytes());  
  
    System.setIn(bais);  
  
    CoffeeMachineUI.main(null);  
  
    String actual = baos.toString();  
  
    assertEquals(expected, actual);  
}  
}
```

## Link of Source Code

[https://drive.google.com/drive/folders/1\\_rvjGqonQF-Iq1jkR9u5EmemIdTtJ5vY](https://drive.google.com/drive/folders/1_rvjGqonQF-Iq1jkR9u5EmemIdTtJ5vY)