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

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

1-CoffeeMachineRefactored

*CoffeeMachineRefactored.java

```java
package CoffeeMachine;

public class CoffeeMachineRefactored
{
    /********* attributes *******/
    private double coffee_powder;
    private double  milk;
    private double water;
    private int Coffee_Count;
    /******** constructor ********/
    public CoffeeMachineRefactored(double coffee_powder, double milk, double water) { // Constructor Initialization...
        this.coffee_powder = coffee_powder;
        this.milk = milk;
        this.water = water;
        this.Coffee_Count = 0;
    }
    /******* getters *******/
    public double getPowder() {return coffee_powder;}
    public double getMilk() {return milk;}
    public double getWater() {return water;}
    public int getCoffee_Count() {return Coffee_Count;}

    /******* setters *******/
    public void setPowder(double coffee_powder) {this.coffee_powder = coffee_powder;}
    public void setMilk(double milk) {this.milk =  milk;}
    public void setWater(double water) {this.water =  water;}

    /******* fundamental functions *******/
    public void SetIngredient(){   //Calling for Filling Ingredient...
        this.coffee_powder = 500.0;
        this.milk = 1;
        this.water= 2;
    }
    public void CleanMachine(){    //Initialization with Null In order to Clean Machine
        this.coffee_powder = 0;
        this.milk = 0;
        this.water= 0;
    }
    public int makecoffee(char t){  //Coffee Selection Menu
        int result = 0;
        boolean available = false;
        switch(t)
        {
            case '1':
                available = this.BlackCoffee();     //Call to BlackCoffee Method
                if(available)
                {
```

*CoffeeMachineRefactored.java

```java
        {
            case '1':
                available = this.BlackCoffee();     //Call to BlackCoffee Method
                if(available)
                {
                    result = 1;
                }
                else
                {
                    result = 0;
                }
                break;
            case '2':
                available = this.MilkCoffee();      //Call to MilkCoffee Method
                if(available)
                {
                    result = 2;
                }
                else
                {
                    result = 0;
                }
                break;
            case '0':
                result = 0;
                break;
            default:
                result = t - '0';
                break;
        }
        return result;
    }
    public boolean BlackCoffee(){
        if(this.coffee_powder >= 10 && this.water >= 0.2){
            this.coffee_powder -= 10;
            this.water -= 0.2;
            this.Coffee_Count++;
            return true;
        }
        return false;
    }
    public boolean MilkCoffee(){
        if(this.coffee_powder >= 10 && this.milk >= 0.4 && this.water >= 0.2){
            this.coffee_powder = this.coffee_powder - 10;
            this.milk -= 0.4;
            this.water -= 0.2;
            this.Coffee_Count++;
            return true;
        }
        return false;
    }
}
```

2-CoffeeMachineUI

*CoffeeMachineRefactored.java | *CoffeeMachineUI.java

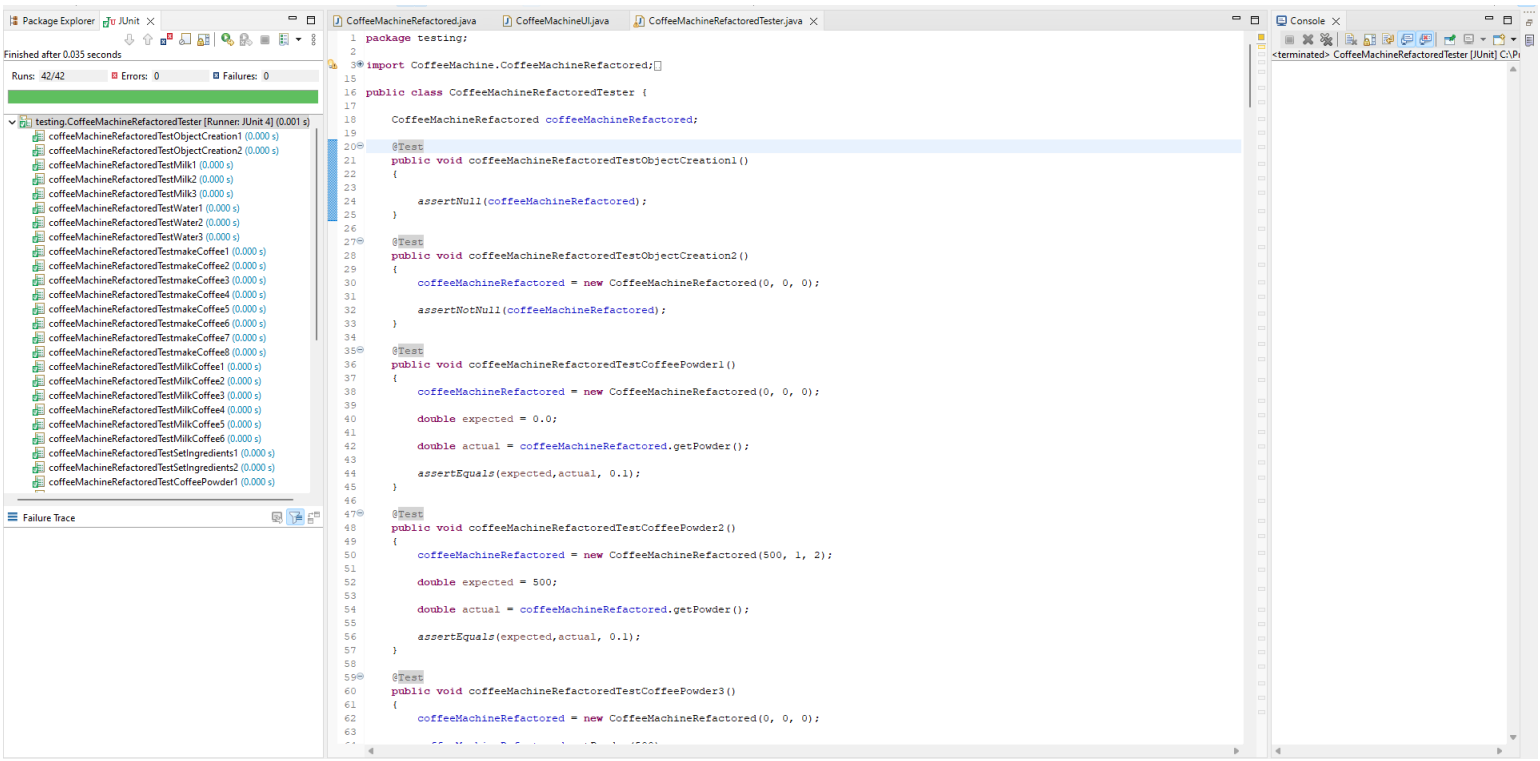
```java
package CoffeeMachine;

import java.util.Scanner;
public class CoffeeMachineUI
{
    /********* attributes *******/
    static Scanner scan;
    CoffeeMachineRefactored CoffeeMachineRefactoredObject;

    /******** constructor *******/
    public CoffeeMachineUI(CoffeeMachineRefactored c)
    {
        scan = new Scanner(System.in);
        this.CoffeeMachineRefactoredObject = c;
    }

    /********* fundamental functions ********/
    private void GetIngredient(){   //To Get Status
        System.out.println("Available Coffee Power(Gram) "+String.format("%.1f",CoffeeMachineRefactoredObject.getPowder()));
        System.out.println("Available Milk(Liter) "+String.format("%.1f", CoffeeMachineRefactoredObject.getMilk()));
        System.out.println("Available Water(Liter) "+String.format("%.1f", CoffeeMachineRefactoredObject.getWater()));
    }

    public void start()
    {    //public Start to accese all private method of this class
        System.out.println(" ----------------------------------------------------------------");
        System.out.println("|                    Coffee Machine By Manikant                    |");
        System.out.println(" ----------------------------------------------------------------");

        System.out.println("\nCurrent Status: ");

        this.GetIngredient();

        boolean t = true;
        while(t){
            System.out.println("\n ------------------------------- ");
            System.out.println("|1:     Status of Ingredient     |\n ------------------------------- \n|2:      Fill Ingredient
            System.out.println(" ------------------------------- \n\n");
            char c = CoffeeMachineUI.scan.next().charAt(0);
            switch(c){
                case '1':
                    System.out.println("------------- Status ------------");
                    this.GetIngredient();
                    System.out.println("----------------------------------");
                    break;
                case '2':
                    System.out.println("\nFilling...");
                    CoffeeMachineRefactoredObject.SetIngredient();
                    System.out.println("Filling Completed.");
                    break;
                case '3':
                    System.out.println("\nCleaning Machine...");
```

*CoffeeMachineRefactored.java *CoffeeMachineUI.java

```java
                    System.out.println("\nCleaning Machine...");
                    CoffeeMachineRefactoredObject.CleanMachine();
                    System.out.println("Cleaning Completed.");
                    break;
                case '4':
                    System.out.println("\n ----------------- ");
                    System.out.println("|   Select Type:   |\n ----------------- \n| 1:  Black Coffee |\n| 2:  Milk Coffee  |\n| 0   to
                    System.out.println(" ----------------- \n");
                    char option = scan.next().charAt(0);
                    int result = CoffeeMachineRefactoredObject.makecoffee(option);
                    if(result == 1)
                    {
                        System.out.println("\nAvailable Coffee Power(Gram) "+String.format("%.1f",CoffeeMachineRefactoredObject.getPowder
                        System.out.println("Available Water(Liter) "+String.format("%.1f", CoffeeMachineRefactoredObject.getWater()));
                    }
                    else if(result == 2)
                    {
                        System.out.println("\nAvailable Coffee Power(Gram) "+String.format("%.1f",CoffeeMachineRefactoredObject.getPowder
                        System.out.println("Available Milk(Liter) "+String.format("%.1f", CoffeeMachineRefactoredObject.getMilk()));
                        System.out.println("Available Water(Liter) "+String.format("%.1f", CoffeeMachineRefactoredObject.getWater()));
                    }
                    else if((option == '1' || option == '2') && result == 0)
                    {
                        System.out.println("\nSome Iteams Are Not Available, Please Fill before Making Coffee.");
                    }
                    else if(option == '0' && result == 0)
                    {
                        //Discard
                    }
                    else
                    {
                        System.out.println("\nBad choice.");
                    }
                    break;
                case '5':
                    System.out.println("\nWe Have Made "+CoffeeMachineRefactoredObject.getCoffee_Count()+" Coffees.");
                    break;
                case '6':
                    System.out.println("\nExiting...\n");
                    t = false;
                    break;
            }
        }
    }

    public static void main(String[] args)
    {
        CoffeeMachineRefactored coffeeMachineObj = new CoffeeMachineRefactored(0,0,0);
        CoffeeMachineUI CoffeeMachineUIObj = new CoffeeMachineUI(coffeeMachineObj);

        System.out.println("\nWants to Start Machine Y or N ?");

        char d;
        d = scan.next().charAt(0);

        if(d == 'Y' || d == 'y')
        {
            CoffeeMachineUIObj.start();     //In order to call All Private Method Calling Public mathod
            System.out.println("Shutting Down...\n");
        }
        else
            System.out.println("Shutting Down...\n");
    }
}
```

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);
    }
```

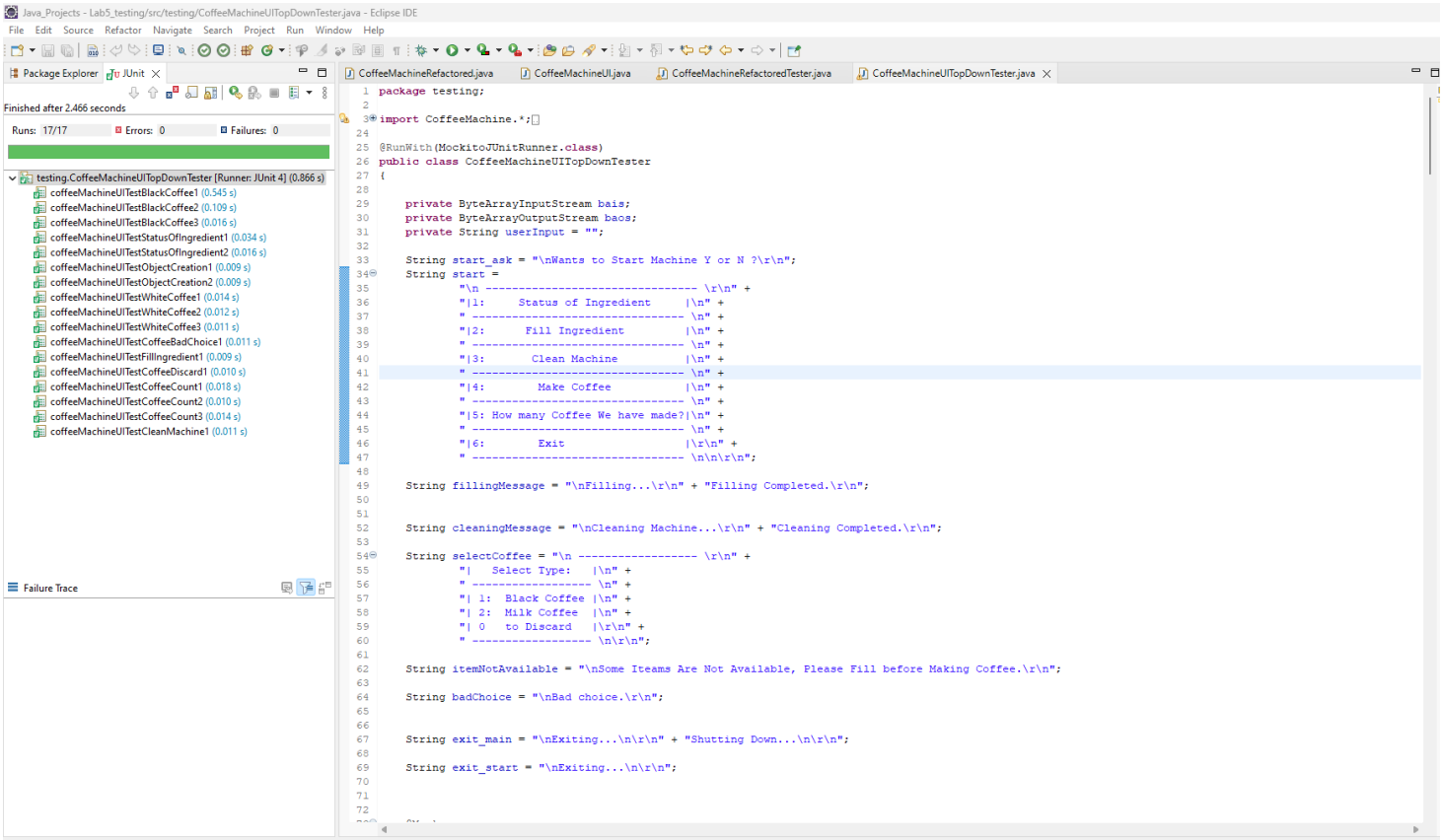
```
@Test
public void coffeeMachineRefactoredTestCoffeeCount1()
{
    coffeeMachineRefactored = new CoffeeMachineRefactored(0, 0, 0);


    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



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

```
@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();
    }
}
```

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     |\n" +
            " ------------------------------- \n" +
            "|2:      Fill Ingredient         |\n" +
            " ------------------------------- \n" +
            "|3:       Clean Machine          |\n" +
            " ------------------------------- \n" +
            "|4:        Make Coffee           |\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 Iteams 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\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());
```

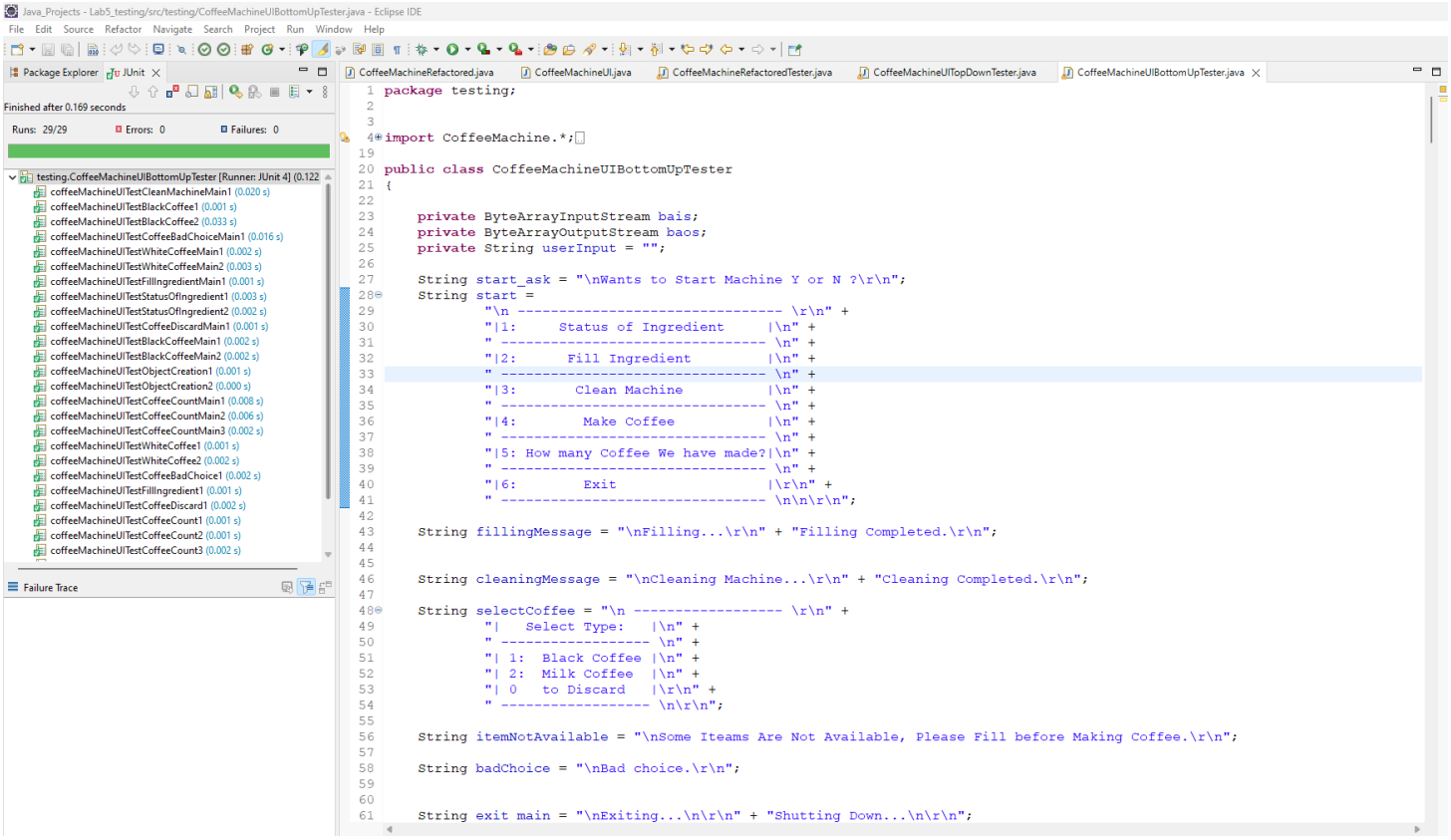
```
        System.setIn(bais);

        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



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     |\n" +
            " ------------------------------- \n" +
            "|2:      Fill Ingredient         |\n" +
            " ------------------------------- \n" +
            "|3:       Clean Machine          |\n" +
            " ------------------------------- \n" +
            "|4:        Make Coffee           |\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 Iteams 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";



    private CoffeeMachineRefactored  coffeeMachineRefactored;

    private CoffeeMachineUI coffeeMachineUI;

    public String GetStartStatus()
    {
        String start_status = " ---------------------------------------------------------------
\r\n" +
                "|                    Coffee Machine By Manikant                    |\r\n" +
                " ---------------------------------------------------------------\r\n\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);;
    }
```

```java
@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