Faculty of Engineering
Ain Shams University

REPORT LAB (3)

NAME	ID	EMAIL
Mennat – Allah Ashraf Fouad Fetouh	**17p3051**	mennatallaashraf@gmail.com

Submitted to:

Dr. Islam El-Maddah
Eng. Adham Nour

Problem 1:

Implement Coffee Machine example with integration testing.

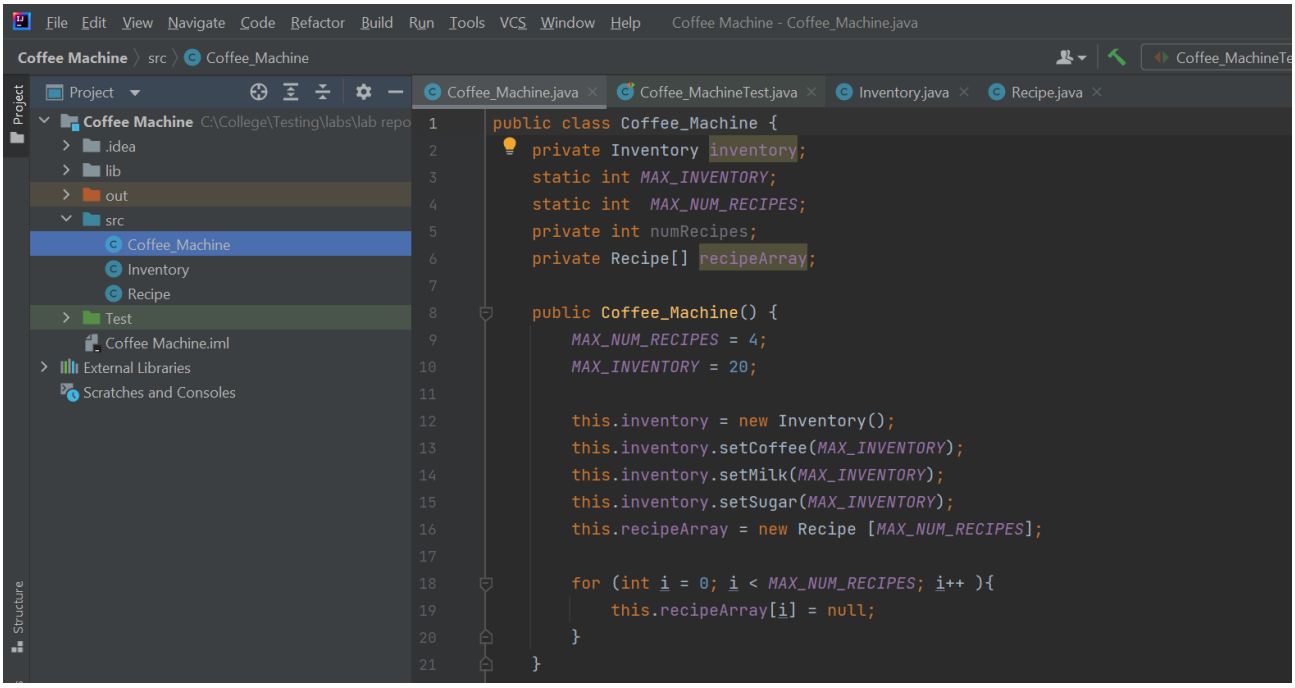
Solution of Problem 1:

1- The main code:

Coffee_Machine.java

```
public class Coffee_Machine {
    private Inventory inventory;
    static int MAX_INVENTORY;
    static int MAX_NUM_RECIPES;
    private int    numRecipes;
    private Recipe[] recipeArray;

    public Coffee_Machine() {
        MAX_NUM_RECIPES = 4;
        MAX_INVENTORY = 20;

        this.inventory = new Inventory();
        this.inventory.setCoffee(MAX_INVENTORY);
        this.inventory.setMilk(MAX_INVENTORY);
        this.inventory.setSugar(MAX_INVENTORY);
        this.recipeArray = new Recipe [MAX_NUM_RECIPES];

        for (int i = 0; i < MAX_NUM_RECIPES; i++ ){
            this.recipeArray[i] = null;
        }
    }

    public boolean addRecipe(Recipe rAdd) {
        for (int i = 0; i < MAX_NUM_RECIPES; i++ ){
            if (this.recipeArray[i] == null) {
                this.recipeArray[i] = rAdd;
                this.numRecipes++;
                return true;
            }
        }
        return false;
    }


 public boolean deleteRecipe(java.lang.String recipeName) {
        for (int i = 0; i < MAX_NUM_RECIPES; i++ ){
            if (this.recipeArray[i].getName() == recipeName) {
                this.recipeArray[i] = null;
                this.numRecipes--;
                return true;
            }

        }
        return false;
    }
```

```java
public Inventory getInventory() {
        return this.inventory;
    }

    public Recipe getRecipe(java.lang.String recipeName) {

        for (int i = 0; i < MAX_NUM_RECIPES; i++ ){
            if (this.recipeArray[i].getName() == recipeName)
                return this.recipeArray[i];
        }
        return null;
    }

    public boolean makeCoffee(java.lang.String recipeName) {
        Recipe currentRecipe = this.getRecipe(recipeName);
        if (currentRecipe != null) {
            this.inventory.setCoffee(this.inventory.getCoffee() -
currentRecipe.getCoffeeLevel());
            this.inventory.setMilk(this.inventory.getMilk() -
currentRecipe.getMilkLevel());
            this.inventory.setSugar(this.inventory.getSugar() -
currentRecipe.getSugarLevel());
            return true;
        }
        return false;
    }
}
```



```java
    public boolean addRecipe(Recipe rAdd) {
        for (int i = 0; i < MAX_NUM_RECIPES; i++ ){
            if (this.recipeArray[i] == null) {
                this.recipeArray[i] = rAdd;
                this.numRecipes++;
                return true;
            }
        }
        return false;
    }

    public boolean deleteRecipe(java.lang.String recipeName) {
        for (int i = 0; i < MAX_NUM_RECIPES; i++ ){
            if (this.recipeArray[i].getName() == recipeName) {
                this.recipeArray[i] = null;
                this.numRecipes--;
                return true;
            }

        }
        return false;
    }
```

```java
    public Inventory getInventory() { return this.inventory; }

    public Recipe getRecipe(java.lang.String recipeName) {

        for (int i = 0; i < MAX_NUM_RECIPES; i++ ){
            if (this.recipeArray[i].getName() == recipeName)
                return this.recipeArray[i];
        }
        return null;
    }

    public boolean makeCoffee(java.lang.String recipeName) {
        Recipe currentRecipe = this.getRecipe(recipeName);
        if (currentRecipe != null) {
            this.inventory.setCoffee(this.inventory.getCoffee() - currentRecipe.getCoffeeLevel());
            this.inventory.setMilk(this.inventory.getMilk() - currentRecipe.getMilkLevel());
            this.inventory.setSugar(this.inventory.getSugar() - currentRecipe.getSugarLevel());
            return true;
        }
        return false;
    }
}
```

Class 2 : Inventory.java

```java
public class Inventory {
    private int     coffee;
    private int     milk;
    private int     sugar;

    public Inventory(){
    }

    public int getCoffee() {
        return this.coffee;
    }
```

```java
    public int getMilk() {
        return this.milk;
    }

    public int getSugar() {
        return this.sugar;
    }

    public void setCoffee(int coffee) {
        this.coffee = coffee;
    }

    public void setMilk(int milk) {
        this.milk = milk;
    }

    public void setSugar(int sugar) {
        this.sugar = sugar;
    }
}
```

Run Tools VCS Window Help Coffee Machine - Inventory.java

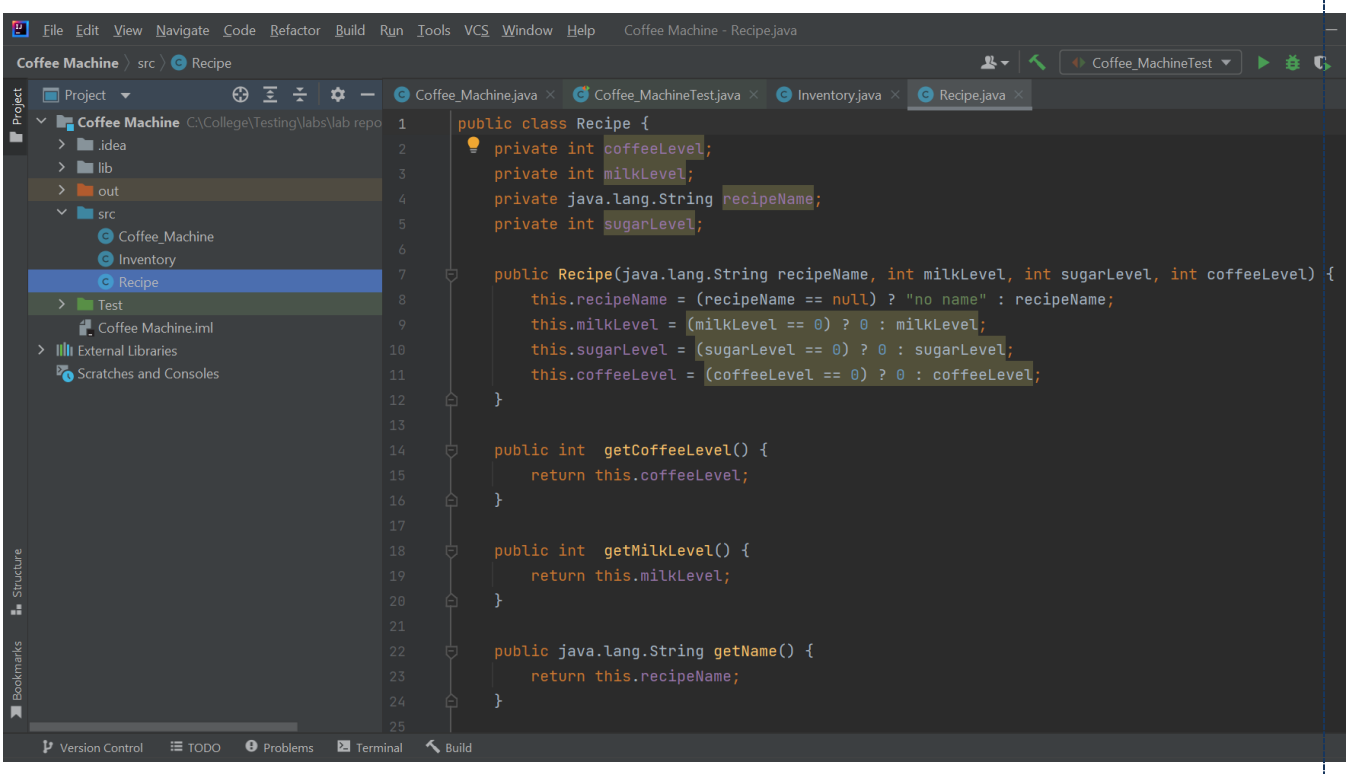
Coffee_Machine.java | Coffee_MachineTest.java | Inventory.java | Recipe.java

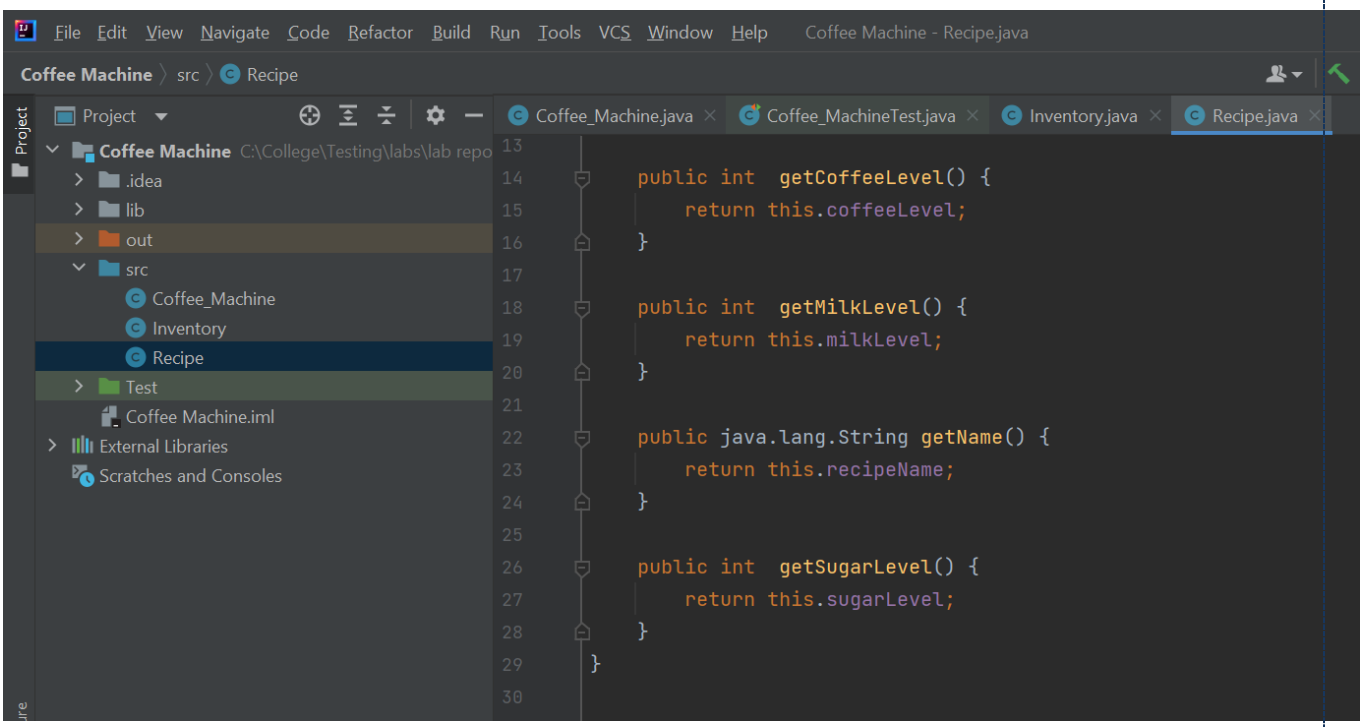
```java
1   public class Inventory {
2       private int coffee;
3       private int milk;
4       private int sugar;
5
6       public Inventory(){
7       }
8
9       public int  getCoffee() { return this.coffee; }
12
13      public int  getMilk() {
14          return this.milk;
15      }
16
17      public int  getSugar() { return this.sugar; }
20
21      public void setCoffee(int coffee) { this.coffee = coffee; }
24
25      public void setMilk(int milk) { this.milk = milk; }
28
29      public void setSugar(int sugar) { this.sugar = sugar; }
32  }
```

inal Build

Class 3 : Recipe.java

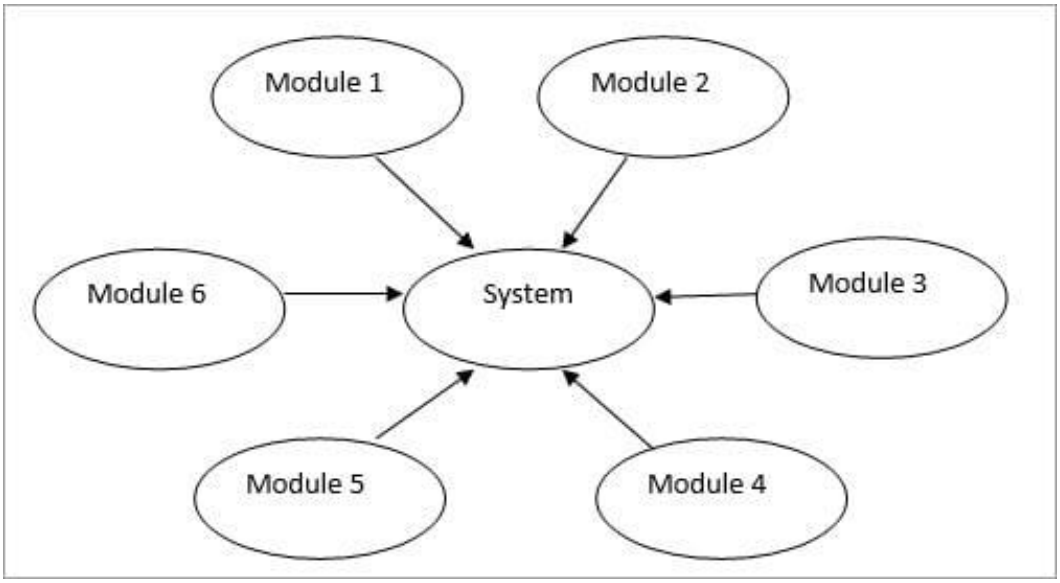
```java
public class Recipe {
    private int    coffeeLevel;
    private int    milkLevel;
    private java.lang.String recipeName;
    private int    sugarLevel;

    public Recipe(java.lang.String recipeName, int milkLevel, int sugarLevel, int
coffeeLevel) {
        this.recipeName = (recipeName == null) ? "no name" : recipeName;
        this.milkLevel = (milkLevel == 0) ? 0 : milkLevel;
        this.sugarLevel = (sugarLevel == 0) ? 0 : sugarLevel;
        this.coffeeLevel = (coffeeLevel == 0) ? 0 : coffeeLevel;
    }

    public int getCoffeeLevel() {
        return this.coffeeLevel;
    }

    public int getMilkLevel() {
        return this.milkLevel;
    }

    public java.lang.String    getName() {
        return this.recipeName;
    }

    public int getSugarLevel() {
        return this.sugarLevel;
    }
}
```





2- Integration Testing:

Using **Big Bang Approach**, which integrates all the modules in one. It does not go for integrating the modules one by one. It verifies if the system works as expected or not once integrated.



Test Cases:

Coffee_MachineTest.java

```java
import org.junit.Test;
import static org.junit.Assert.*;

public class Coffee_MachineTest {

    @Test
    public void addRecipe() {
        java.lang.String recipeName = "Dark Roast";
        int milkLevel = 2;
        int sugarLevel = 3;
        int coffeeLevel = 4;

        Coffee_Machine testCoffeeMaker = new Coffee_Machine();
        Recipe testRecipe = new Recipe(recipeName, milkLevel, sugarLevel, coffeeLevel);

        assertEquals(recipeName, testRecipe.getName());
        assertEquals(coffeeLevel, testRecipe.getCoffeeLevel());
        assertEquals(sugarLevel, testRecipe.getSugarLevel());
        assertEquals(milkLevel, testRecipe.getMilkLevel());

        if (!testCoffeeMaker.addRecipe(testRecipe)) {
            fail("Recipe not Added");
        }

        assertNotNull(testCoffeeMaker.getRecipe(recipeName));
    }

    @Test
    public void makeCoffee() {
        java.lang.String recipeName = "Dark Roast";
        int milkLevel = 2;
        int sugarLevel = 3;
        int coffeeLevel = 4;

        Coffee_Machine testCoffeeMaker = new Coffee_Machine();
        Recipe testRecipe = new Recipe(recipeName, milkLevel, sugarLevel, coffeeLevel);
```

```
        if (!testCoffeeMaker.addRecipe(testRecipe))
            fail("Recipe not Added");

        if(!testCoffeeMaker.makeCoffee(testRecipe.getName()))
            fail("Coffee not Made");

        // Ensures inventory is updated
        assertEquals(testCoffeeMaker.MAX_INVENTORY - coffeeLevel,
testCoffeeMaker.getInventory().getCoffee());
        assertEquals(testCoffeeMaker.MAX_INVENTORY - sugarLevel,
testCoffeeMaker.getInventory().getSugar());
        assertEquals(testCoffeeMaker.MAX_INVENTORY - milkLevel,
testCoffeeMaker.getInventory().getMilk());
    }

    @Test
    public void deleteRecipe() {
        java.lang.String recipeName = "Dark Roast";
        int milkLevel = 2;
        int sugarLevel = 3;
        int coffeeLevel = 4;

        Coffee_Machine testCoffeeMaker = new Coffee_Machine();
        Recipe testRecipe = new Recipe(recipeName, milkLevel, sugarLevel, coffeeLevel);

        if (!testCoffeeMaker.addRecipe(testRecipe))
            fail("Recipe not Added");
        if (!testCoffeeMaker.deleteRecipe(testRecipe.getName()))
            fail("Recipe not Deleted");
    }
```

```java
import org.junit.Test;
import static org.junit.Assert.*;

public class Coffee_MachineTest {

    @Test
    public void addRecipe() {
        java.lang.String recipeName = "Dark Roast";
        int milkLevel = 2;
        int sugarLevel = 3;
        int coffeeLevel = 4;

        Coffee_Machine testCoffeeMaker = new Coffee_Machine();
        Recipe testRecipe = new Recipe(recipeName, milkLevel, sugarLevel, coffeeLevel);

        assertEquals(recipeName, testRecipe.getName());
        assertEquals(coffeeLevel, testRecipe.getCoffeeLevel());
        assertEquals(sugarLevel, testRecipe.getSugarLevel());
        assertEquals(milkLevel, testRecipe.getMilkLevel());

        if (!testCoffeeMaker.addRecipe(testRecipe)) {
            fail("Recipe not Added");
        }
        assertNotNull(testCoffeeMaker.getRecipe(recipeName));
    }
```

```java
    @Test
    public void makeCoffee() {
        java.lang.String recipeName = "Dark Roast";
        int milkLevel = 2;
        int sugarLevel = 3;
        int coffeeLevel = 4;

        Coffee_Machine testCoffeeMaker = new Coffee_Machine();
        Recipe testRecipe = new Recipe(recipeName, milkLevel, sugarLevel, coffeeLevel);

        if (!testCoffeeMaker.addRecipe(testRecipe))
            fail("Recipe not Added");

        if(!testCoffeeMaker.makeCoffee(testRecipe.getName()))
            fail("Coffee not Made");

        // Ensures inventory is updated
        assertEquals( expected: testCoffeeMaker.MAX_INVENTORY - coffeeLevel, testCoffeeMaker.getInventory().getCoffee());
        assertEquals( expected: testCoffeeMaker.MAX_INVENTORY - sugarLevel, testCoffeeMaker.getInventory().getSugar());
        assertEquals( expected: testCoffeeMaker.MAX_INVENTORY - milkLevel, testCoffeeMaker.getInventory().getMilk());
    }
```

```java

    @Test
    public void deleteRecipe() {
        java.lang.String recipeName = "Dark Roast";
        int milkLevel = 2;
        int sugarLevel = 3;
        int coffeeLevel = 4;

        Coffee_Machine testCoffeeMaker = new Coffee_Machine();
        Recipe testRecipe = new Recipe(recipeName, milkLevel, sugarLevel, coffeeLevel);

        if (!testCoffeeMaker.addRecipe(testRecipe))
            fail("Recipe not Added");
        if (!testCoffeeMaker.deleteRecipe(testRecipe.getName()))
            fail("Recipe not Deleted");
    }

}
```

Test Cases "Run" window:

```java
import org.junit.Test;
import static org.junit.Assert.*;

public class Coffee_MachineTest {

    @Test
    public void addRecipe() {
        java.lang.String recipeName = "Dark Roast";
        int milkLevel = 2;
        int sugarLevel = 3;
        int coffeeLevel = 4;

        Coffee_Machine testCoffeeMaker = new Coffee_Machine();
        Recipe testRecipe = new Recipe(recipeName, milkLevel, sugarLevel, coffeeLevel);
```

```
Run: Coffee_MachineTest
Tests passed: 3 of 3 tests – 33 ms
Coffee_MachineTest 33 ms
    deleteRecipe 32 ms
    makeCoffee 0 ms
    addRecipe 1 ms

"C:\Program Files\Java\jdk1.8.0_111\bin\java.exe" ...

Process finished with exit code 0
```

Problem 2:

Implement ATM Machine example with integration testing.

Solution of Problem 2:

1- The main code:

ATM.java

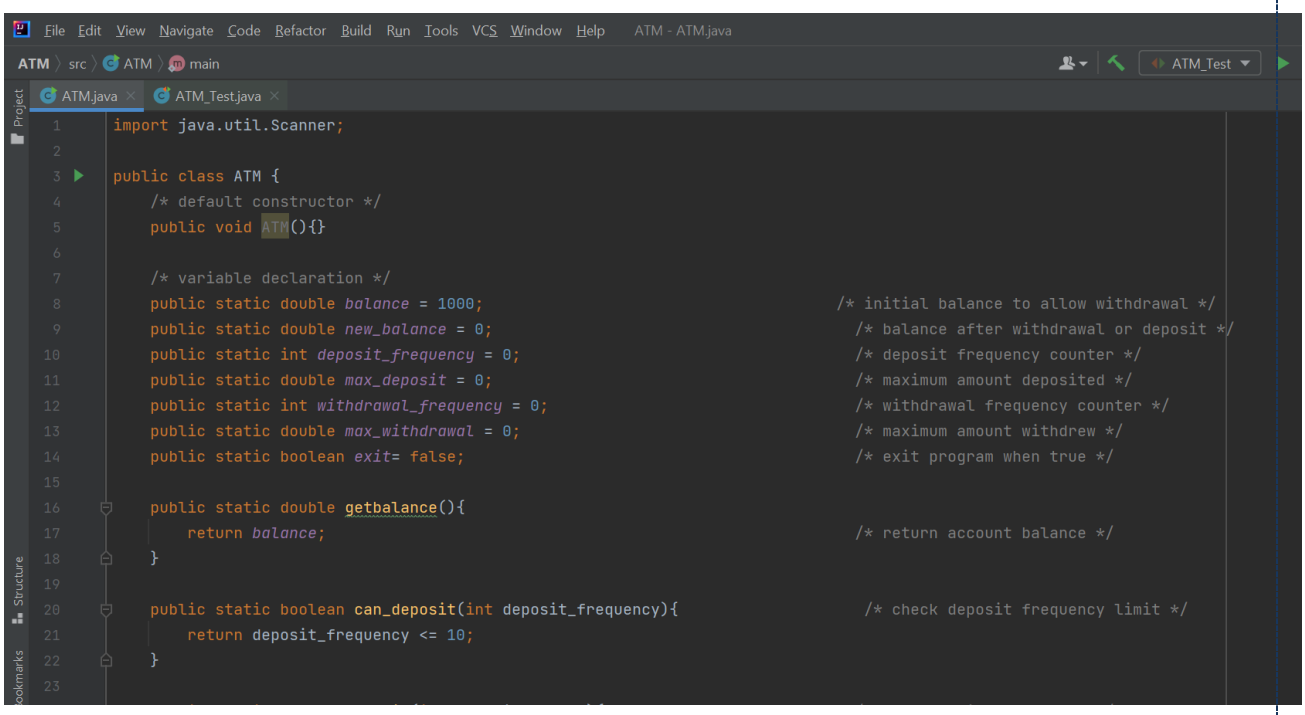
```
import java.util.Scanner;

public class ATM {
    /* default constructor */
    public void ATM(){}

    /* variable declaration */
    public static double balance = 1000;
/* initial balance to allow withdrawal */
    public static double new_balance = 0;
/* balance after withdrawal or deposit */
    public static int deposit_frequency = 0;
/* deposit frequency counter */
    public static double max_deposit = 0;
/* maximum amount deposited */
    public static int withdrawal_frequency = 0;
/* withdrawal frequency counter */
    public static double max_withdrawal = 0;
/* maximum amount withdrew */
    public static boolean exit= false;
/* exit program when true */

    public static double getbalance(){
        return balance;
/* return account balance */
    }

    public static boolean can_deposit(int deposit_frequency){
/* check deposit frequency limit */
        return deposit_frequency <= 10;
    }

    public static double deposit(int deposit_amount){
/* add deposit to balance */

        if(deposit_amount > 0){
/* run only when deposit amount is more than zero */

            if(deposit_amount <= 10000){
/* deposit amount must be at most 10000 */
                max_deposit += deposit_amount;
```

```
                if(max_deposit <= 100000){
/* maximum deposit for the day must be at most 100000 */
                    System.out.printf("Current Balance: %.2f", balance);
                    System.out.println("");
                    balance += deposit_amount;
                    return balance;
                }
                else {
                    System.out.println("ERROR: You have reached the limit for
maximum deposit per day.");
                }
            }

            else {
                System.out.println("ERROR: Maximum deposit amount at most
10000 L.E.");
            }

        } else {
            System.out.println("ERROR: Enter a valid amount.");
        }

        return balance;

    }

    public static boolean can_withdraw(int withdrawal_frequency){
/* check withdrawal frequency limit */
        return withdrawal_frequency <= 5;
    }

    public static double withdrawal(int withdrawal_amount){
/* subtract withdrawal from balance */

        if(withdrawal_amount > 0){
/* run only if withdrawal amount is more than zero */

            if(withdrawal_amount < balance){
/* withdrawal amount must not be more than balance */

                if(withdrawal_amount <= 10000){
/* withdrawal amount must be at most 10000 */
                    max_withdrawal += withdrawal_amount;

                    if(max_withdrawal <= 50000){
/* maximum withdrawal for the day must be at most 50000 */
                        System.out.printf("Current Balance: %.2f", balance);
                        System.out.println("");
                        balance -= withdrawal_amount;
                        return balance;
                    }

                    else {
```

```
                        System.out.println("ERROR: You have reached the limit
for maximum withdrawal per day.");
                    }
                }

                else {
                    System.out.println("ERROR: Maximum withdrawal amount must
be at most 10000 L.E.");
                }
            }

            else {

                System.out.printf("ERROR: Withdrawal amount is more than the
balance: %.2f", balance);
                System.out.println("");

            }
        }

        else {
            System.out.println("ERROR: Enter a valid amount.");
        }

        return balance;
    }

    public static boolean quit(String answer){
/* exit program */

        if(answer.equals("yes") || answer.equals("YES")){
/* exit only after confirmation from user */
            return true;
        }

        else {
            return false;
        }

    }

    public static boolean userInput(String choice){

        switch(choice){
            case "BALANCE": System.out.printf("%.2f", getbalance());
/* user wants to check balance */
                System.out.println("");
                System.out.println("Type MENU to go back");
                String menu = new Scanner(System.in).nextLine();
                break;

            case "DEPOSIT": deposit_frequency++;
                if(can_deposit(deposit_frequency)){
```

```
/* allow only if maximum frequency not met */
                    System.out.print("Please enter the amount to deposit: ");
                    int deposit_amount = new Scanner(System.in).nextInt();
                    new_balance = deposit(deposit_amount);                  /*
user wants to deposit money */
                    System.out.printf("New Balance: %.2f", new_balance);
                    System.out.println("");
                }

                else {                                                      /*
maximum frequency for the day met */
                    System.out.println("ERROR: You have reached the maximum
frequency of deposited per day.");
                }
                break;

            case "WITHDRAWAL": withdrawal_frequency++;
                if(can_withdraw(withdrawal_frequency)){
/* allow only if maximum frequency not met */
                    System.out.print("Please enter the amount to withdraw: ");
                    int withdrawal_amount = new Scanner(System.in).nextInt();
                    new_balance = withdrawal(withdrawal_amount);            /*
user wants to withdraw money */
                    System.out.printf("New Balance: %.2f", new_balance);
                    System.out.println("");
                }

                else {                                                      /*
maximum frequncy met for the day */
                    System.out.println("ERROR: You have reach the maximum
frequency of withdrawal per day.");
                }
                break;

            case "EXIT": System.out.print("Are you sure you want to
exit?(YES/NO): ");
                String answer = new Scanner(System.in).nextLine();
                if(quit(answer)){                                           /*
user wants to quit */
                        /*System.exit(0);
                        System.gc();*/
                    return true;
                }
                break;

            default: System.out.println("ERROR: Enter from available
options");
                break;

        }
        return false;
    }
```

```java
    public static void main(String[] args) {

        System.out.println("Welcome to the ATM Program");

        while(!exit){

            System.out.println("Below are the options for you to choose
from:\n" +
                    "1. Type BALANCE to check your Balance\n" +
                    "2. Type DEPOSIT to deposit money\n" +
                    "3. Type WITHDRAWAL to withdraw money\n" +
                    "4. Type EXIT to exit the program");            /* Ask
user for input */

            Scanner scan = new Scanner(System.in);
            String c = scan.nextLine();

            exit = userInput(c);
/* process user input */
        }
        System.exit(0);
        System.gc();
    }
}
```



```java
public static double deposit(int deposit_amount){                        /* add deposit to balance */

    if(deposit_amount > 0){                                              /* run only when deposit amount is more than zero */

        if(deposit_amount <= 10000){                                     /* deposit amount must be at most 10000 */
            max_deposit += deposit_amount;
            if(max_deposit <= 100000){                                   /* maximum deposit for the day must be at most 100000 */
                System.out.printf( s: "Current Balance: %.2f", balance);
                System.out.println("");
                balance += deposit_amount;
                return balance;
            }
            else {
                System.out.println("ERROR: You have reached the limit for maximum deposit per day.");
            }
        }

        else {
            System.out.println("ERROR: Maximum deposit amount at most 10000 L.E.");
        }

    } else {
        System.out.println("ERROR: Enter a valid amount.");
    }
```

```java
    return balance;

}

public static boolean can_withdraw(int withdrawal_frequency){            /* check withdrawal frequency limit */
    return withdrawal_frequency <= 5;
}

public static double withdrawal(int withdrawal_amount){                  /* subtract withdrawal from balance */

    if(withdrawal_amount > 0){                                           /* run only if withdrawal amount is more than zero */

        if(withdrawal_amount < balance){                                 /* withdrawal amount must not be more than balance */

            if(withdrawal_amount <= 10000){                              /* withdrawal amount must be at most 10000 */
                max_withdrawal += withdrawal_amount;

                if(max_withdrawal <= 50000){                             /* maximum withdrawal for the day must be at most 50000 */
                    System.out.printf( s: "Current Balance: %.2f", balance);
                    System.out.println("");
                    balance -= withdrawal_amount;
                    return balance;
                }
```

```java
                else {
                    System.out.println("ERROR: You have reached the limit for maximum withdrawal per day.");
                }
            }

            else {
                System.out.println("ERROR: Maximum withdrawal amount must be at most 10000 L.E.");
            }
        }

        else {

            System.out.printf( s: "ERROR: Withdrawal amount is more than the balance: %.2f", balance);
            System.out.println("");

        }
    }

    else {
        System.out.println("ERROR: Enter a valid amount.");
    }

    return balance;
}
```

```java
public static boolean quit(String answer){                          /* exit program */

    if(answer.equals("yes") || answer.equals("YES")){               /* exit only after confirmation from user */
        return true;
    }

    else {
        return false;
    }

}

public static boolean userInput(String choice){

    switch(choice){
        case "BALANCE": System.out.printf( s: "%.2f", getbalance());        /* user wants to check balance */
            System.out.println("");
            System.out.println("Type MENU to go back");
            String menu = new Scanner(System.in).nextLine();
            break;

        case "DEPOSIT": deposit_frequency++;
            if(can_deposit(deposit_frequency)){                     /* allow only if maximum frequency not met */
                System.out.print("Please enter the amount to deposit: ");
                int deposit_amount = new Scanner(System.in).nextInt();
```

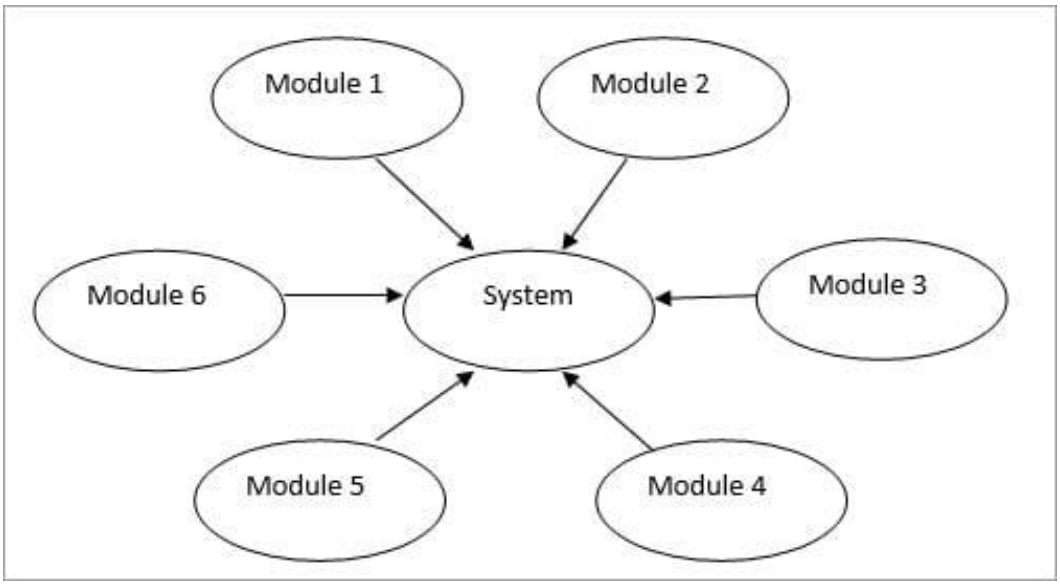
```java
                int deposit_amount = new Scanner(System.in).nextInt();
                new_balance = deposit(deposit_amount);                  /* user wants to deposit money */
                System.out.printf( s: "New Balance: %.2f", new_balance);
                System.out.println("");
            }

            else {                                                      /* maximum frequency for the day met */
                System.out.println("ERROR: You have reached the maximum frequency of deposited per day.");
            }
            break;

        case "WITHDRAWAL": withdrawal_frequency++;
            if(can_withdraw(withdrawal_frequency)){                     /* allow only if maximum frequency not met */
                System.out.print("Please enter the amount to withdraw: ");
                int withdrawal_amount = new Scanner(System.in).nextInt();
                new_balance = withdrawal(withdrawal_amount);            /* user wants to withdraw money */
                System.out.printf( s: "New Balance: %.2f", new_balance);
                System.out.println("");
            }

            else {                                                      /* maximum frequncy met for the day */
                System.out.println("ERROR: You have reach the maximum frequency of withdrawal per day.");
            }
            break;
```

```java
            else {                                                      /* maximum frequncy met for the day */
                System.out.println("ERROR: You have reach the maximum frequency of withdrawal per day.");
            }
            break;

        case "EXIT": System.out.print("Are you sure you want to exit?(YES/NO): ");
            String answer = new Scanner(System.in).nextLine();
            if(quit(answer)){                                           /* user wants to quit */
                    /*System.exit(0);
                    System.gc();*/
                return true;
            }
            break;

        default: System.out.println("ERROR: Enter from available options");
            break;

    }
    return false;
}
```

```java
    public static void main(String[] args) {

        System.out.println("Welcome to the ATM Program");

        while(!exit){

            System.out.println("Below are the options for you to choose from:\n" +
                    "1. Type BALANCE to check your Balance\n" +
                    "2. Type DEPOSIT to deposit money\n" +
                    "3. Type WITHDRAWAL to withdraw money\n" +
                    "4. Type EXIT to exit the program");            /* Ask user for input */

            Scanner scan = new Scanner(System.in);
            String c = scan.nextLine();

            exit = userInput(c);                                        /* process user input */
        }
        System.exit( 0);
        System.gc();
    }
}
```

2- Integration Testing:

Using **Big Bang Approach**, which integrates all the modules in one. It does not go for integrating the modules one by one. It verifies if the system works as expected or not once integrated.



Test Cases:

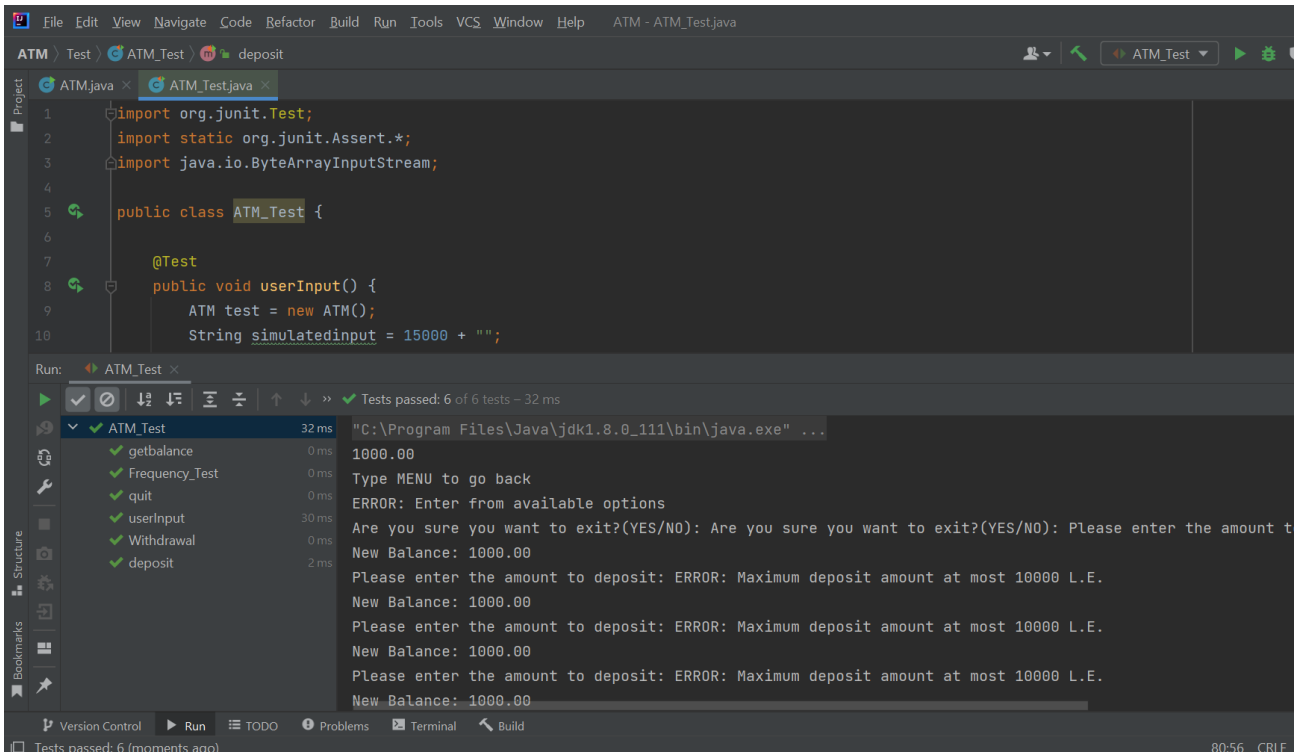
ATM_Test.java

```java
import org.junit.Test;
import static org.junit.Assert.*;
import java.io.ByteArrayInputStream;

public class ATM_Test {

    @Test
    public void userInput() {
        ATM test = new ATM();
        String simulatedinput = 15000 + "";
        ByteArrayInputStream in = new ByteArrayInputStream("MENU".getBytes());
        System.setIn(in);
        assertFalse(test.userInput("BALANCE"));
        assertFalse(test.userInput("DEFAULT"));
        in = new ByteArrayInputStream("YES".getBytes());
        System.setIn(in);
        assertTrue(test.userInput("EXIT"));
        in = new ByteArrayInputStream("NO".getBytes());
        System.setIn(in);
        assertFalse(test.userInput("EXIT"));
        in = new ByteArrayInputStream(simulatedinput.getBytes());
        System.setIn(in);
        assertFalse(test.userInput("DEPOSIT"));
        in = new ByteArrayInputStream(simulatedinput.getBytes());
        System.setIn(in);
        assertFalse(test.userInput("DEPOSIT"));
        in = new ByteArrayInputStream(simulatedinput.getBytes());
        System.setIn(in);
        assertFalse(test.userInput("DEPOSIT"));
        in = new ByteArrayInputStream(simulatedinput.getBytes());
        System.setIn(in);
        assertFalse(test.userInput("DEPOSIT"));
        in = new ByteArrayInputStream(simulatedinput.getBytes());
        System.setIn(in);
        assertFalse(test.userInput("DEPOSIT"));
        in = new ByteArrayInputStream(simulatedinput.getBytes());
        System.setIn(in);
```

```java
        assertFalse(test.userInput("WITHDRAWAL"));
        in = new ByteArrayInputStream(simulatedinput.getBytes());
        System.setIn(in);
        assertFalse(test.userInput("WITHDRAWAL"));
        in = new ByteArrayInputStream(simulatedinput.getBytes());
        System.setIn(in);
        assertFalse(test.userInput("WITHDRAWAL"));
        in = new ByteArrayInputStream(simulatedinput.getBytes());
        System.setIn(in);
        assertFalse(test.userInput("WITHDRAWAL"));
    }

    @Test
    public void getbalance() {
        ATM test = new ATM();
        assertEquals(1000, test.getbalance(), 0);
    }

    @Test
    public void Frequency_Test(){
        ATM test = new ATM();
        assertTrue(test.can_deposit(3));
        assertFalse(test.can_deposit(11));
        assertTrue(test.can_withdraw(2));
        assertFalse(test.can_withdraw(6));
    }
@Test
public void Withdrawal(){
    ATM test = new ATM();
    assertEquals("ERORR",1000, test.withdrawal(10000), 0);
    assertEquals("ERORR",1000, test.withdrawal(-50), 0);
    assertEquals(500, test.withdrawal(500), 0);

}

    @Test
    public void deposit() {
        ATM test = new ATM();
assertEquals(10500, test.deposit(10000), 0);
assertEquals("ERROR: Maximum deposit amount at most 10000 L.E.",10500, test.deposit(40000), 0);
assertEquals(18000, test.deposit(7500), 0);
assertEquals("ERROR: Enter a valid amount.",18000, test.deposit(-50), 0);    }
```

```java
    @Test
    public void quit() {
        ATM test = new ATM();
        assertFalse(test.quit("NO"));
        assertTrue(test.quit("YES"));
        assertTrue(test.quit("yes"));
    }

}
```

Test Cases "Run" window:



Problem 3:

Implement Digital Watch example with integration testing.

DigitalWatch.java

```
public class DigitalWatch {
    public String DigitalWatch ( String input ){
        int len = input.length();
        int m=0 ,h = 0;
        int D = 1,M = 1;
        int Y = 2000;
        String s;
        String state = "Normal_Display";
        String i1 = "Time";
        String i2 = "Alarm";
        String i3 = "min";

        for(int i=0; i< len; i++){
            switch (state){

                case "Normal_Display" : {
                    if ( input.charAt(i) == 'c' )
                        state = "Update";
                    if ( input.charAt(i) == 'b' )
                        state = "Alarm_Set";
                    if ( input.charAt(i) == 'a' ){
                        if( i1 == "Time")
                            i1 = "Date";
                        else
                            i1 = "Time";
                    }
                    break;
                }

                case "Alarm_Set" : {
                    if ( input.charAt(i) == 'a' ) {
                        if (i2 == "Alarm")
```

```
                i2 = "Alarm_Set";
            }
            if ( input.charAt(i) == 'd' )
                state = "Normal_Display";
            break;
        }

        case "Update" : {
            if (input.charAt(i) == 'a') {
                if (i3 == "min")
                    i3 = "hour";
                else if (i3 == "hour")
                    i3 = "day";
                else if (i3 == "day")
                    i3 = "month";
                else if (i3 == "month")
                    i3 = "year";
                else if (i3 == "year")
                    state = "Normal_Display";
            }

            if (input.charAt(i) == 'b'){
                if (i3 == "min") {
                    if (m < 60)
                        m++;
                    else
                        m=0;
                }
                else if (i3 == "hour")
                    if ( h < 24)
                        h++;
                    else
                        h=0;
                else if (i3 == "day")
                    if ( D < 31)
                        D++;
                    else
                        D=1;
                else if (i3 == "month")
                    if ( M < 12)
                        M++;
                    else
```

```
                        M=1;
                    else if (i3 == "year")
                        if ( Y < 2100)
                            Y++;
                }

                if (input.charAt(i) == 'd')
                    state = "Normal_Display";
                break;
            }
        }
    }

    if (state == "Normal_Display") {
        s = i1;
    }
    else if (state == "Alarm_Set"){
        s = i1;
    }
    else {
        s = i1;
    }

    return "Current state: " + state + ", the inner state: " + s + "  Date: " +
            String.valueOf(D) + " - " + String.valueOf(M) + " - " +String.valueOf(Y) +
            "  Time: " + String.format("%02d", h) + ":" + String.format("%02d", m);
  }
}
```

2-Integration Testing:

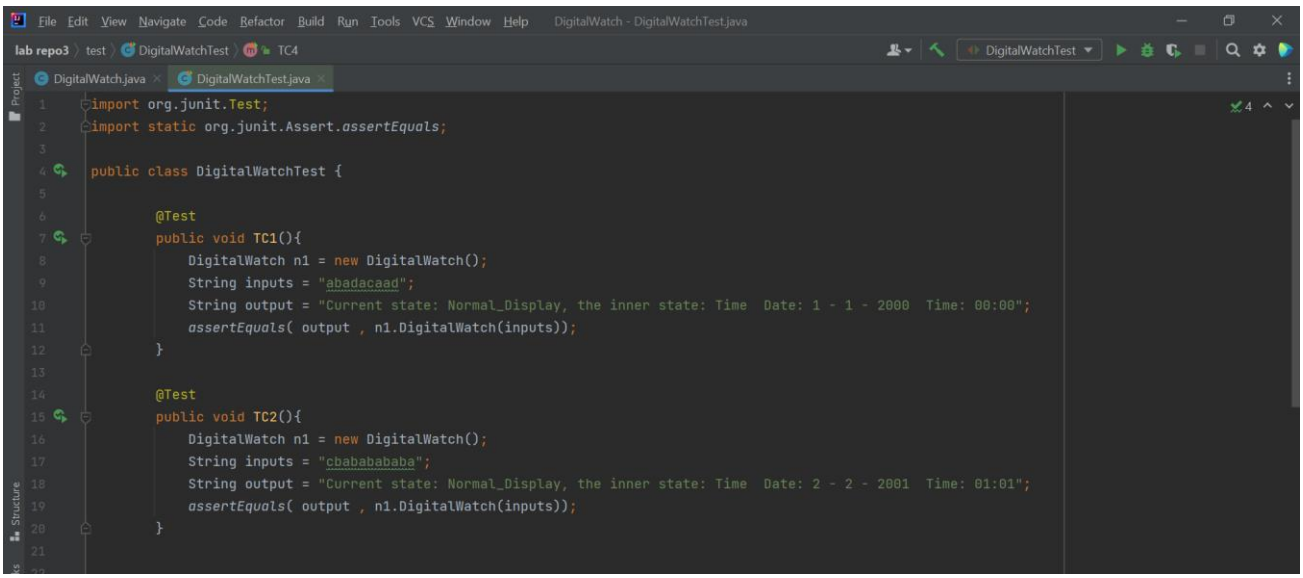
Using **Big Bang Approach**, which integrates all the modules in one. It does not go for integrating the modules one by one. It verifies if the system works as expected or not once integrated.

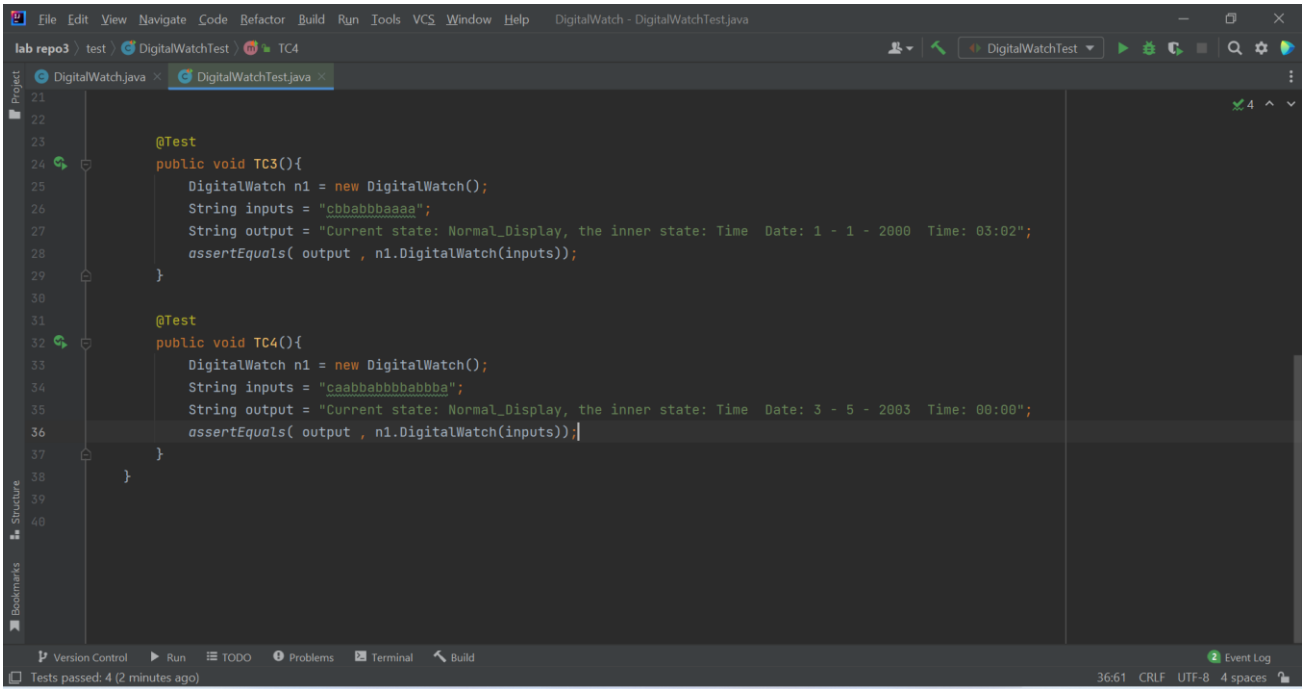
Test Cases:

DigitalWatchTest.java

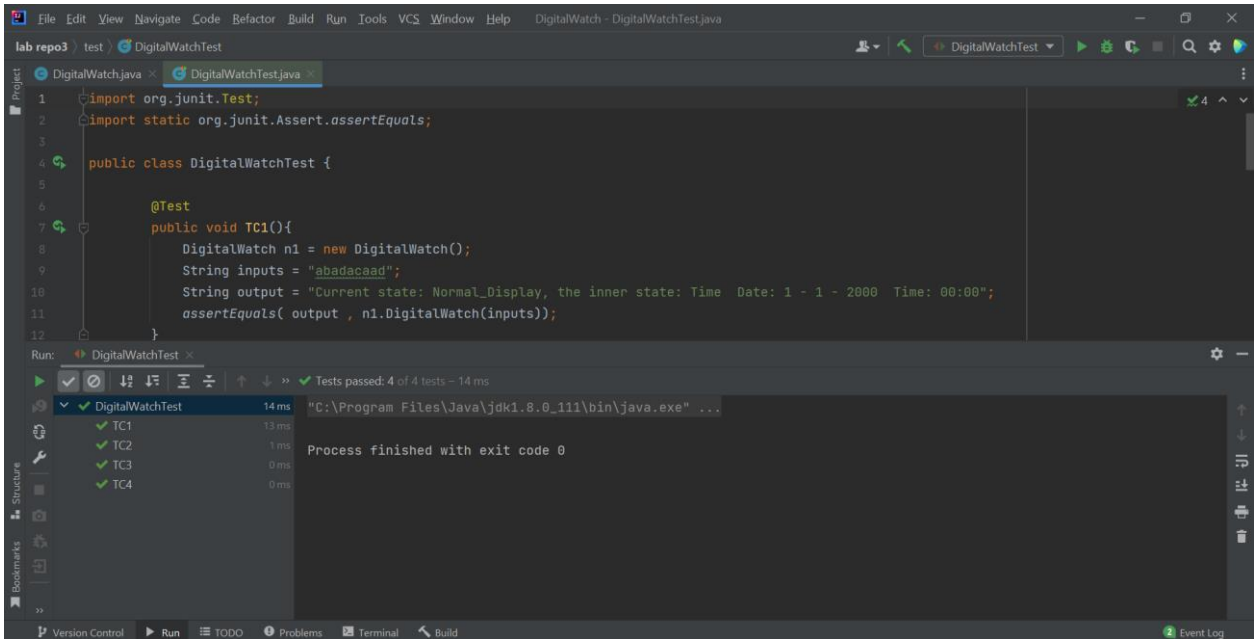
```
import org.junit.Test;
import static org.junit.Assert.assertEquals;

public class DigitalWatchTest {

        @Test
        public void TC1(){
            DigitalWatch n1 = new DigitalWatch();
            String inputs = "abadacaad";
            String output = "Current state: Normal_Display, the inner state: Time  Date: 1 - 1 -
2000  Time: 00:00";
            assertEquals( output , n1.DigitalWatch(inputs));
        }

        @Test
        public void TC2(){
            DigitalWatch n1 = new DigitalWatch();
            String inputs = "cbabababa";
            String output = "Current state: Normal_Display, the inner state: Time  Date: 2 - 2 -
2001  Time: 01:01";
            assertEquals( output , n1.DigitalWatch(inputs));
        }


        @Test
        public void TC3(){
            DigitalWatch n1 = new DigitalWatch();
            String inputs = "cbbabbbaaaa";
            String output = "Current state: Normal_Display, the inner state: Time  Date: 1 - 1 -
2000  Time: 03:02";
            assertEquals( output , n1.DigitalWatch(inputs));
        }

        @Test
```

```
        public void TC4(){
            DigitalWatch n1 = new DigitalWatch();
            String inputs = "caabbabbbbabbba";
            String output = "Current state: Normal_Display, the inner state: Time  Date: 3 - 5 -
2003  Time: 00:00";
            assertEquals( output , n1.DigitalWatch(inputs));
        }
    }
```





Test Cases "Run" window:



GitHub Repo:

GitHub Repo Link is:

https://github.com/Mennah-Ashraf/Testing-Lab3-Repo.git