JUnit Assignment

File=First.java

```java
package com.im;

import java.lang.reflect.Array;
import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.IntStream;
import java.util.stream.Stream;

public class First {

  public static void main(String[] args) {

    First first = new First();
    //System.out.println(first.replaceSubString("This is my main string text", "main", "modified"));
    //System.out.println(first.filterEvenElements(IntStream.of(1,2,3,4,5).boxed().collect(Collectors.toList())));
    BigDecimal result = first.calculateAverage(new ArrayList<>());
  }

  /**
   * This method is used to replace the part of string with new string.
   * @param mainString string which needs to be modified
   * @param subString string which needs to be replaced
   * @param  replacementString string to be replaced with
   * @return updated string if mainString contains substring, else original string
   */
  public String replaceSubString(String mainString, String subString, String replacementString) {
    if(!mainString.isEmpty() && subString != null && replacementString != null && mainString.contains(subString)) {
      return mainString.replaceAll(subString, replacementString);
    }else {
      return mainString;
    }
  }

  /**
   * This method is used to filter even elements from list.
   * @param list list of integer
   * @return list
   */
  public List<Integer> filterEvenElements(List<Integer> list) {
    Iterator<Integer> it = list.iterator();
    while(it.hasNext()) {
      if(it.next() % 2 == 0) {
        it.remove();
```

```
        }
      }
      return list;
    }

    public BigDecimal calculateAverage(List<BigDecimal> values) {
      if (values == null || values.size() < 1) {
        throw new RuntimeException("Invalid input");
      } else {
        BigDecimal sum = values.stream().reduce(BigDecimal.ZERO, BigDecimal::add);
        return (sum.divide(new BigDecimal(values.size())));
      }
    }

    public Boolean isPallindrome(String origString) {
      Boolean isPallindrome = false;
      String reverseString = new StringBuilder(origString).reverse().toString();

      // Check palindrome string
      if (origString.equals(reverseString)) {
        isPallindrome = true;
      }
      return isPallindrome;
    }
}
```

File=FirstTest.java

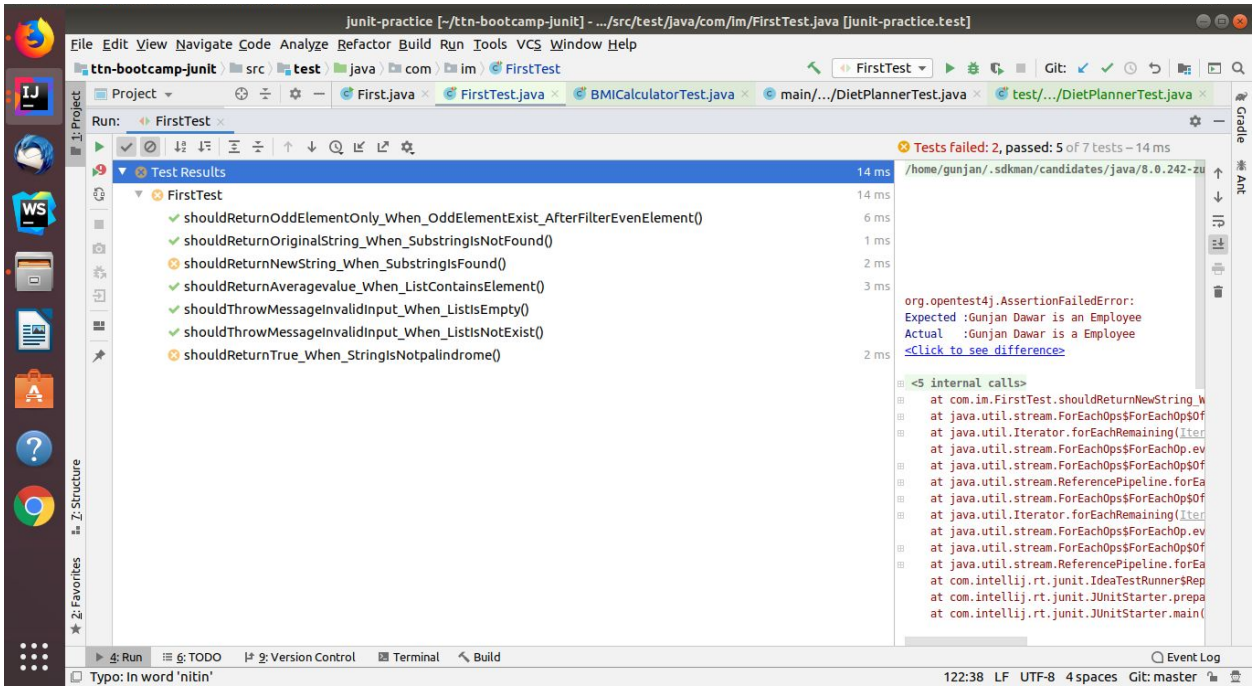
```
package com.im;

import org.junit.jupiter.api.Test;

import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.List;

import static org.junit.jupiter.api.Assertions.*;

public class FirstTest {
    First first = new First();
    @Test
    void shouldReturnNewString_When_SubstringIsFound() {
      //given
      String mainString = "Gunjan Dawar is a Trainee";
      String findstring = "Trainee";
      String replacedString = "Employee";
      String expectedString = "Gunjan Dawar is an Employee";
```

```
    //when
    String calculateString = first.replaceSubString(mainString, findstring, replacedString);

    //then
    assertEquals(expectedString, calculateString);
}

@Test
void shouldReturnOriginalString_When_SubstringIsNotFound() {
    //given
    String mainString = "Gunjan kathuria";
    String findstring = "kathuria";
    String replacedString = "Dawar";
    String expectedString = "Gunjan Dawar";

    //when
    String calculateString = first.replaceSubString(mainString, findstring, replacedString);

    //then
    assertEquals(expectedString, calculateString);

}

@Test
void shouldReturnOddElementOnly_When_OddElementExist_AfterFilterEvenElement() {
    //given
    List<Integer> list = new ArrayList<>();
    for (int i = 1; i < 6; i++) {
        list.add(i);
    }
    List<Integer> expectedlist = new ArrayList<>();
    expectedlist.add(1);
    expectedlist.add(3);
    expectedlist.add(5);

    //when
    List calculatelist = first.filterEvenElements(list);

    //then
    assertEquals(expectedlist, calculatelist);

}
```

```
@Test
void shouldThrowMessageInvalidInput_When_ListIsNotExist() {
    //given
    List<BigDecimal> list = null;
    //List<BigDecimal> list=new ArrayList<>();

    //when
    try {
        first.calculateAverage(list);
    }

    //then
    catch (RuntimeException r) {
        System.out.println(r);

    }
}

@Test
void shouldThrowMessageInvalidInput_When_ListIsEmpty() {
    //given
    List<BigDecimal> list = new ArrayList<>();

    //when
    try {
        first.calculateAverage(list);
    }

    //then
    catch (RuntimeException r) {
        System.out.println(r);

    }
}

@Test
void shouldReturnAveragevalue_When_ListContainsElement() {

    //given
    List<BigDecimal> list = new ArrayList<>();
    for (int i = 0; i < 4; i++) {
        list.add(new BigDecimal(1212121));
        list.add(new BigDecimal(1212121));
        list.add(new BigDecimal(1212121));
        list.add(new BigDecimal(1212121));
    }
    BigDecimal expectedaverage = new BigDecimal(1212121);

    //when
    BigDecimal calculateaverage = first.calculateAverage(list);
```

```
    //then
    assertEquals(expectedaverage, calculateaverage);
  }

  @Test
  void shouldReturnTrue_When_StringIsNotpalindrome() {
    //given
    String originalinput = "nitin";

    //when
    boolean palindromecheck = first.isPallindrome(originalinput);

    //then
    assertFalse(palindromecheck);

  }
}
```

OUTPUT:



File=BMICalculator.java

```java
package healthycoderapp;

import java.util.Comparator;
import java.util.List;

public class BMICalculator {

  private static final double BMI_THRESHOLD = 25.0;

  public static boolean isDietRecommended(double weight, double height) {
    if (height == 0.0) throw new ArithmeticException();
    double bmi = weight / (height * height);
    if (bmi < BMI_THRESHOLD)
      return false;
    return true;
  }

  public static Coder findCoderWithWorstBMI(List<Coder> coders) {
    return coders.stream().sorted(Comparator.comparing(BMICalculator::calculateBMI))
        .reduce((first, second) -> second).orElse(null);
  }

  public static double[] getBMIScores(List<Coder> coders) {
    double[] bmiScores = new double[coders.size()];
    for (int i = 0; i < bmiScores.length; i++) {
      bmiScores[i] = BMICalculator.calculateBMI(coders.get(i));
    }
    return bmiScores;
  }

  private static double calculateBMI(Coder coder) {
    double height = coder.getHeight();
    double weight = coder.getWeight();
    if (height == 0.0)
      throw new ArithmeticException();
    double bmi = weight / (height * height);
    return Math.round(bmi * 100) / 100.0;
  }

}
```

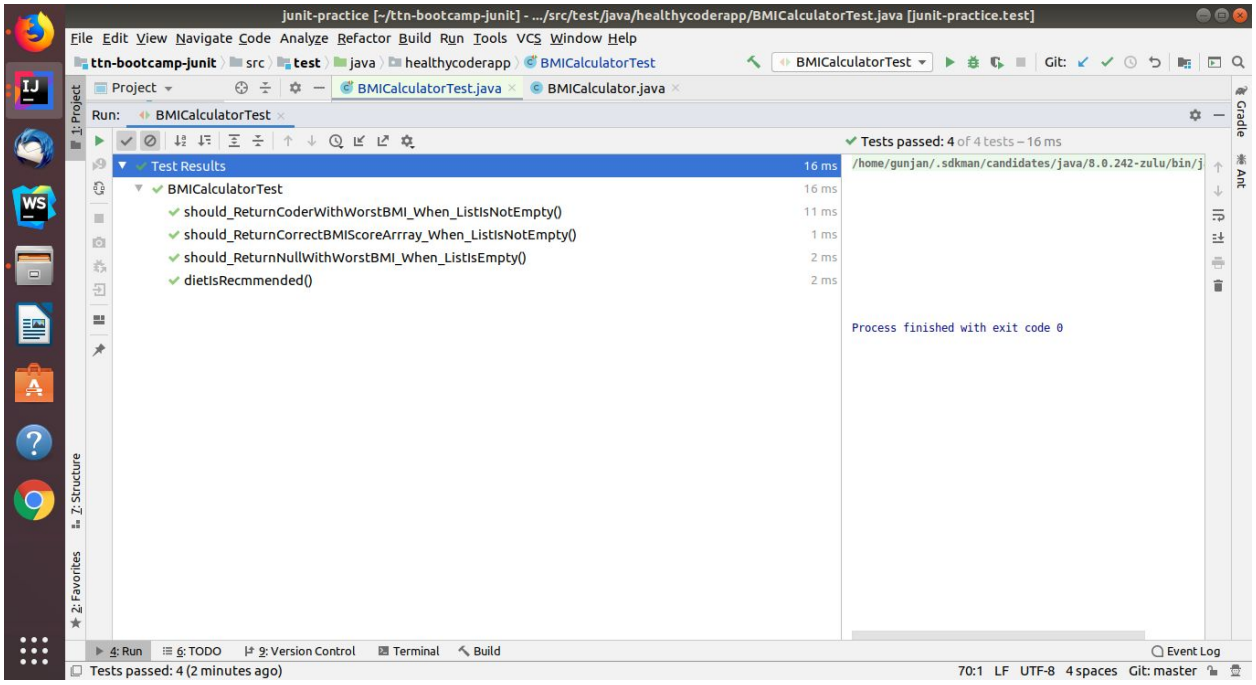
File=BMICalculatorTest.java

```java
package healthycoderapp;

import org.junit.jupiter.api.Test;

import java.util.ArrayList;
import java.util.List;
```

```java
import static org.junit.jupiter.api.Assertions.*;

class BMICalculatorTest {

    @Test
    void dietIsRecmmended(){
        assertTrue(true);

        //given
        double weight=90.0;
        double height=1.72;

        //when
        boolean recomend=BMICalculator.isDietRecommended(weight,height);

        //then
        assertTrue(recomend);

    }

    @Test
    void should_ReturnCoderWithWorstBMI_When_ListIsNotEmpty()
    {
        //given
        List<Coder> list=new ArrayList<>();
        list.add(new Coder(1.80,60));
        list.add(new Coder(1.82,98.0));
        list.add(new Coder(1.82,64.7));

        //When
        Coder codercalculate=BMICalculator.findCoderWithWorstBMI(list);

        //then
        assertAll(
                ()->assertEquals(1.82,codercalculate.getHeight()),
                ()->assertEquals(98.0,codercalculate.getWeight())

        );

    }
    @Test
    void should_ReturnNullWithWorstBMI_When_ListIsEmpty()
    {
        //given
        List<Coder> list=new ArrayList<>();

        //When
        Coder codercalculate=BMICalculator.findCoderWithWorstBMI(list);
```

```
    //then
    assertNull(codercalculate);
  }

  @Test
  void should_ReturnCorrectBMIScoreArrray_When_ListIsNotEmpty()
  {
    //given
    List<Coder> list=new ArrayList<>();
    list.add(new Coder(1.80,60));
    list.add(new Coder(1.82,98.0));
    list.add(new Coder(1.82,64.7));
    double []expected={18.52 ,29.59 ,19.53};

    //When
    double []bmiScores=BMICalculator.getBMIScores(list);

    //then
    assertArrayEquals(expected,bmiScores);
  }
}
```

OUTPUT:



File=DietPlanner.java

```
package healthycoderapp;

public class DietPlanner {

  private int proteinPercentage;
  private int fatPercentage;
  private int carbohydratePercentage;

  //todo
  public DietPlanner(int proteinPercentage, int fatPercentage, int carbohydratePercentage) {
    super();
    if (proteinPercentage + fatPercentage + carbohydratePercentage != 100) {
      throw new RuntimeException("protein, fat and carbohydrate percentages must add up to 100!");
    }

    this.proteinPercentage = proteinPercentage;
    this.fatPercentage = fatPercentage;
    this.carbohydratePercentage = carbohydratePercentage;
  }

  //todo
  public DietPlan calculateDiet(Coder coder) {
    int calories = this.calculateBMR(coder);
    int protein = this.calculateProtein(calories);
    int fat = this.calculateFat(calories);
    int carbohydrate = this.calculateCarbohydrate(calories);

    return new DietPlan(calories, protein, fat, carbohydrate);
  }

  private int calculateProtein(int bmr) {
    return (int) Math.round(bmr * proteinPercentage / 400.0);
  }
  private int calculateFat(int bmr) {
    return (int) Math.round(bmr * fatPercentage / 900.0);
  }
  private int calculateCarbohydrate(int bmr) {
    return (int) Math.round(bmr * carbohydratePercentage / 400.0);
  }

  private int calculateBMR(Coder coder) {
    if (coder.getGender() == Gender.MALE) {
      return (int) Math.round(
          (66.5 + 13.8 * coder.getWeight()
          + 5.0 * coder.getHeight() * 100
          - 6.8 * coder.getAge()) * 1.2
      );
    }
```

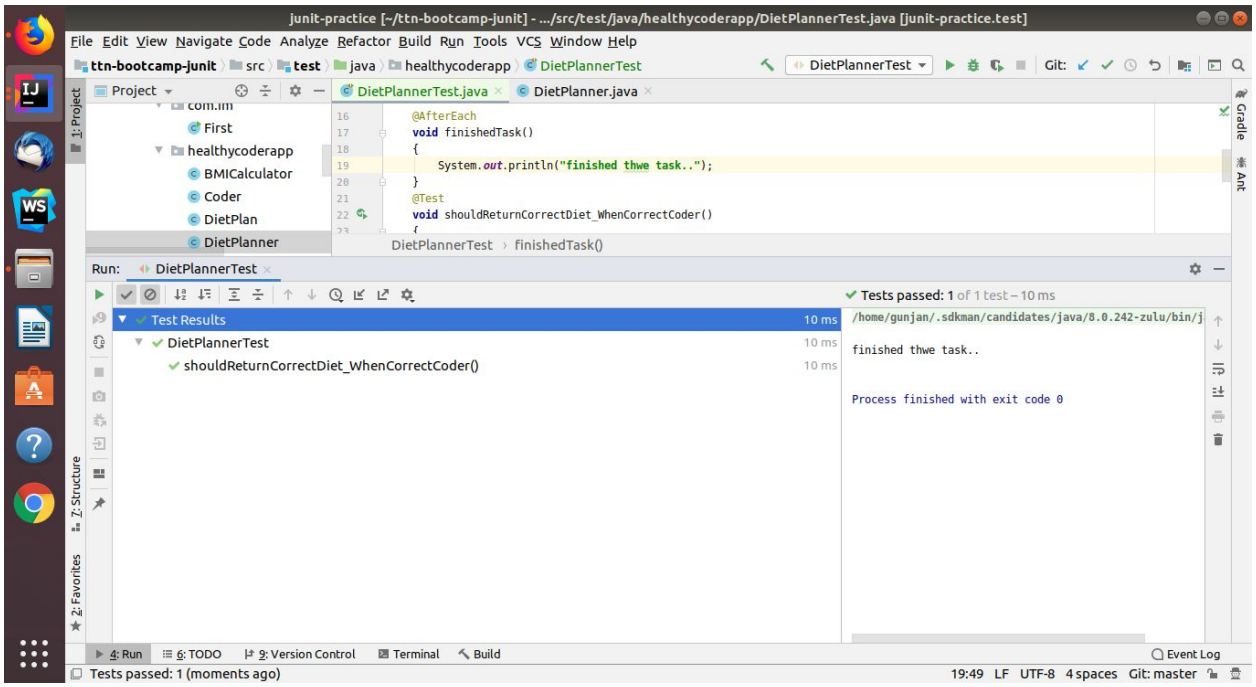
```
    return (int) Math.round(
        (655.1 + 9.6 * coder.getWeight()
        + 1.9 * coder.getHeight() * 100
        - 4.7 * coder.getAge()) * 1.2
    );
  }
}
```

File=DietPlannerTest.java

```
package healthycoderapp;

import org.junit.jupiter.api.*;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;


public class DietPlannerTest {
  private DietPlanner dietPlanner;
  @BeforeEach
  void setup()
  {
     this.dietPlanner=new DietPlanner(20,30,50);
  }

  @AfterEach
  void finishedTask()
  {
     System.out.println("Junit Finished");
  }
  @Test
  void shouldReturnCorrectDiet_WhenCorrectCoder()
  {
     //given
     Coder coder=new Coder(1.82,75,26,Gender.MALE);
     DietPlan dietPlanexpected=new DietPlan(2202,110,73,275);

     //when
     DietPlan actual=dietPlanner.calculateDiet(coder);

     //then
     assertAll(
          ()->assertEquals(dietPlanexpected.getCalories(),actual.getCalories()),
          ()->assertEquals(dietPlanexpected.getProtein(),actual.getProtein()),
          ()->assertEquals(dietPlanexpected.getFat(),actual.getFat()),
          ()->assertEquals(dietPlanexpected.getCarbohydrate(),actual.getCarbohydrate())

     );
```

```
    }
}
```

OUTPUT:


junit-practice [~/ttn-bootcamp-junit] - .../src/test/java/healthycoderapp/DietPlannerTest.java [junit-practice.test]
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
ttn-bootcamp-junit src test java healthycoderapp DietPlannerTest
DietPlannerTest
Git:
Project
DietPlannerTest.java
DietPlanner.java
First
healthycoderapp
BMICalculator
Coder
DietPlan
DietPlanner
@AfterEach
void finishedTask()
{
System.out.println("finished thwe task..");
}
@Test
void shouldReturnCorrectDiet_WhenCorrectCoder()
DietPlannerTest › finishedTask()
Run:
DietPlannerTest
Tests passed: 1 of 1 test – 10 ms
Test Results 10 ms
DietPlannerTest 10 ms
shouldReturnCorrectDiet_WhenCorrectCoder() 10 ms
/home/gunjan/.sdkman/candidates/java/8.0.242-zulu/bin/j
finished thwe task..
Process finished with exit code 0
4: Run
6: TODO
9: Version Control
Terminal
Build
Event Log
Tests passed: 1 (moments ago)
19:49 LF UTF-8 4 spaces Git: master
1: Project
7: Structure
2: Favorites
Gradle
Ant