Q1.*Write all possible (including failure, exception case) Unit Tests for all the methods in First.java.*

Ans.

Code:-

```
package com.im;

//import org.junit.Test;

import junit.runner.Version;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.function.Executable;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.CsvSource;
import org.junit.jupiter.params.provider.ValueSource;
import org.junit.runners.Parameterized;

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

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

public class FirstTest {

/*void canary() {
assertTrue(true);

System.out.println("Test Executed.....");


}*/

@Test
void should_check_for_average() {
List<BigDecimal> list = new ArrayList<BigDecimal>();
list.add(new BigDecimal(10.333));
list.add(new BigDecimal(10.333));
list.add(new BigDecimal(10.333));
list.add(new BigDecimal(10.333));
list.add(new BigDecimal(10.333));

BigDecimal result = new First().calculateAverage(list);
BigDecimal expected = new BigDecimal(10.333);

assertEquals(result, expected);

}
```

```
@Test
   void should_check_for_null_exception() { List<BigDecimal> list =
      new ArrayList<BigDecimal>(); Executable executable = () ->
      new First().calculateAverage(list);

assertThrows(RuntimeException.class, executable);

}

//@ParameterizedTest
//@ValueSource(String ={"Nitin","tushar","Mahak"})
@ParameterizedTest
@CsvSource(value = {"tushar", "Nitin", "Nikina"})
void check_for_palindrome(String val) {
String str = val;
boolean actual = new First().isPallindrome(str);

assertFalse(actual);
//System.out.println("JUnit version is: " + Version.id());
}

@Test
void check_for_even_element() {
List<Integer> ls = new ArrayList<Integer>();
ls.add(234);
ls.add(239);
ls.add(238);
ls.add(237);
ls.add(236);

List<Integer> expected = Arrays.asList(239, 237); List<Integer>
      actual = new First().filterEvenElements(ls); Integer[] exp =
      expected.stream().toArray(Integer[]::new); Integer[] act =
      actual.stream().toArray(Integer[]::new); //for(Integer
      e:actual)

      //            System.out.println("values:
      "+e); assertArrayEquals(exp, act);
```

```
}

@Test
void check_for_null_String() {
String str = "";
boolean expected = true;
boolean actual = new First().isPallindrome(str);

assertEquals(expected, actual);

}

@Test
void check_for_Replace_substring(){
String main = “tushar";
String sub = "i";
String tar = "ii";
String expected = "tushar";
String actual = new First().replaceSubString(main,sub,tar);
      assertEquals(expected,actual);

}

}
```

Q2.*Write Unit tests for HealthyCoder app given in the Udemy session. You need to write tests for the BMICalculator and DitePlanner.*

Ans.

BMICalculato

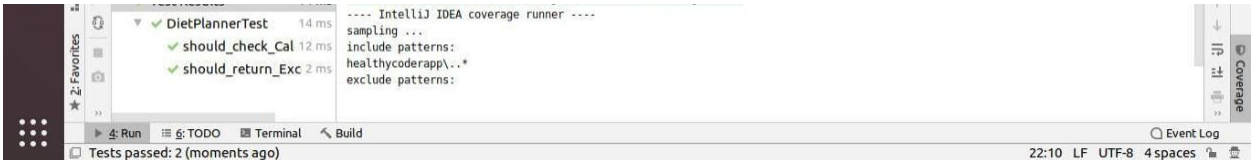
Code:-

```
package healthycoderapp;
import junit.runner.Version;

import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.function.Executable;

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

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

class BMICalculatorTest {

@BeforeAll
static void beforeAll(){
System.out.println("initialization....");

}

@AfterAll
static void afterAll(){
System.out.println("Ending....");

}

void should_Return_True(){
```

```
//given
double weight=89.0;
double height=1.72;

//when
boolean recommend=BMICalculator.isDietRecommended(weight,height);
      //assertTrue(BMICalculator.isDietRecommended(129.0,1.75));

//then
assertTrue(recommend);

}

@Test
void should_Return_False(){

//given
double weight = 50.0;
double height = 1.92;

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

//then

assertFalse(rec);
}

@Test
void should_return_coder_with_worstBMI(){

List<Coder> coders = new ArrayList<Coder>();
coders.add(new Coder(1.80,60.0));
coders.add(new Coder(1.82,98.0));
coders.add(new Coder(1.82,97.0));

Coder coderWorstBMI = BMICalculator.findCoderWithWorstBMI(coders);

assertAll(
() -> assertEquals(1.82, coderWorstBMI.getHeight()),
() -> assertEquals(98.0, coderWorstBMI.getWeight())

);

}

@Test
void should_Check_Exceptions(){

double height = 0;
```

```
double weight = 89.0;

Executable executable = () -> BMICalculator.isDietRecommended(weight,height);

assertThrows(ArithmeticException.class, executable);

}



}
```

DietPlanner:-



Code:-

```
package healthycoderapp;

import org.junit.Assert;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.function.Executable;

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

public class DietPlannerTest {

@Test
void should_return_Exception(){
//Coder coder = new Coder(1.76,78,23,Gender.MALE);

//DietPlan actual = new DietPlanner(30,30,30).calculateDiet(coder);
```

```
Executable executable = () -> new DietPlanner(30,30,30);
assertThrows(RuntimeException.class,executable);

}

@Test
void should_check_Calculate_Diet(){
Coder coder = new Coder(1.76,78,23,Gender.MALE);

DietPlan actual = new DietPlanner(40,30,30).calculateDiet(coder);
      DietPlan expected = new DietPlan(2240,224,75,168); assertAll(

() -> assertEquals(actual.getCalories(), expected.getCalories() ),
() -> assertEquals(actual.getCarbohydrate(), expected.getCarbohydrate() ),
() -> assertEquals(actual.getFat(), expected.getFat() ),
() -> assertEquals(actual.getProtein(), expected.getProtein() )

);

}

}
```

Q1.*Write all possible (including failure, exception case) Unit Tests for all the methods in First.java.*

Ans.

Code:-

```
package com.im;

//import org.junit.Test;

import junit.runner.Version;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.function.Executable;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.CsvSource;
import org.junit.jupiter.params.provider.ValueSource;
import org.junit.runners.Parameterized;

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

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

public class FirstTest {

/*void canary() {
assertTrue(true);
```

```
System.out.println("Test Executed.....");


}*/

@Test
void should_check_for_average() {
List<BigDecimal> list = new ArrayList<BigDecimal>();
list.add(new BigDecimal(10.333));
list.add(new BigDecimal(10.333));
list.add(new BigDecimal(10.333));
list.add(new BigDecimal(10.333));
list.add(new BigDecimal(10.333));

BigDecimal result = new First().calculateAverage(list);
BigDecimal expected = new BigDecimal(10.333);

assertEquals(result, expected);

}

@Test
   void should_check_for_null_exception() { List<BigDecimal> list =
       new ArrayList<BigDecimal>(); Executable executable = () ->
       new First().calculateAverage(list);

assertThrows(RuntimeException.class, executable);

}

//@ParameterizedTest
//@ValueSource(String ={"Nitin","Nikhil","Mahak"})
@ParameterizedTest
@CsvSource(value = {"tusharl", "Nitin", "Nikina"})
void check_for_palindrome(String val) {
String str = val;
boolean actual = new First().isPallindrome(str);

assertFalse(actual);
//System.out.println("JUnit version is: " + Version.id());
}

@Test
void check_for_even_element() {
List<Integer> ls = new ArrayList<Integer>();
ls.add(234);
ls.add(239);
ls.add(238);
ls.add(237);
ls.add(236);

List<Integer> expected = Arrays.asList(239, 237); List<Integer>
       actual = new First().filterEvenElements(ls); Integer[] exp =
       expected.stream().toArray(Integer[]::new); Integer[] act =
       actual.stream().toArray(Integer[]::new); //for(Integer
       e:actual)

      //            System.out.println("values:
       "+e); assertArrayEquals(exp, act);
```

```
}

@Test
void check_for_null_String() {
String str = "";
boolean expected = true;
boolean actual = new First().isPallindrome(str);

assertEquals(expected, actual);

}

@Test
void check_for_Replace_substring(){
String main = "tushar";
String sub = "i";
String tar = "ii";
String expected = "tushar";
String actual = new First().replaceSubString(main,sub,tar);
      assertEquals(expected,actual);

}

}
```

Q2.*Write Unit tests for HealthyCoder app given in the Udemy session. You need to write tests for the BMICalculator and DitePlanner.*

Ans.

BMICalculato

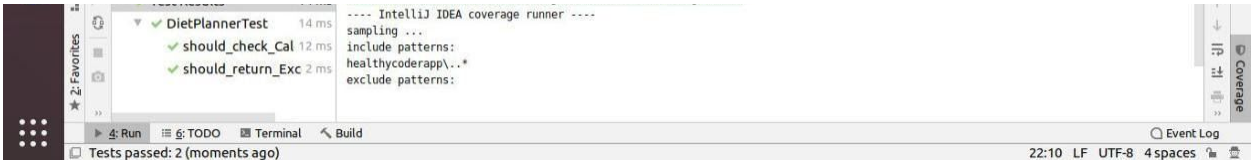
Code:-

```
package healthycoderapp;
import junit.runner.Version;

import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.function.Executable;

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

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

class BMICalculatorTest {

@BeforeAll
static void beforeAll(){
System.out.println("initialization....");

}

@AfterAll
static void afterAll(){
System.out.println("Ending....");

}

void should_Return_True(){
```

```
//given
double weight=89.0;
double height=1.72;

//when
boolean recommend=BMICalculator.isDietRecommended(weight,height);
      //assertTrue(BMICalculator.isDietRecommended(129.0,1.75));

//then
assertTrue(recommend);

}

@Test
void should_Return_False(){

//given
double weight = 50.0;
double height = 1.92;

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

//then

assertFalse(rec);
}

@Test
void should_return_coder_with_worstBMI(){

List<Coder> coders = new ArrayList<Coder>();
coders.add(new Coder(1.80,60.0));
coders.add(new Coder(1.82,98.0));
coders.add(new Coder(1.82,97.0));

Coder coderWorstBMI = BMICalculator.findCoderWithWorstBMI(coders);

assertAll(
() -> assertEquals(1.82, coderWorstBMI.getHeight()),
() -> assertEquals(98.0, coderWorstBMI.getWeight())

);

}

@Test
void should_Check_Exceptions(){

double height = 0;
```

```
double weight = 89.0;

Executable executable = () -> BMICalculator.isDietRecommended(weight,height);

assertThrows(ArithmeticException.class, executable);

}



}
```

DietPlanner:-



Code:-

```
package healthycoderapp;

import org.junit.Assert;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.function.Executable;

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

public class DietPlannerTest {

@Test
void should_return_Exception(){
//Coder coder = new Coder(1.76,78,23,Gender.MALE);

//DietPlan actual = new DietPlanner(30,30,30).calculateDiet(coder);
```

```
Executable executable = () -> new DietPlanner(30,30,30);
assertThrows(RuntimeException.class,executable);

}

@Test
void should_check_Calculate_Diet(){
Coder coder = new Coder(1.76,78,23,Gender.MALE);

DietPlan actual = new DietPlanner(40,30,30).calculateDiet(coder);
      DietPlan expected = new DietPlan(2240,224,75,168); assertAll(

() -> assertEquals(actual.getCalories(), expected.getCalories() ),
() -> assertEquals(actual.getCarbohydrate(), expected.getCarbohydrate() ),
() -> assertEquals(actual.getFat(), expected.getFat() ),
() -> assertEquals(actual.getProtein(), expected.getProtein() )

);

}

}
```