UNIVERSITI TUNKU ABDUL RAHMAN

ACADEMIC YEAR 2019/2020

JANUARY 2020 TRIMESTER

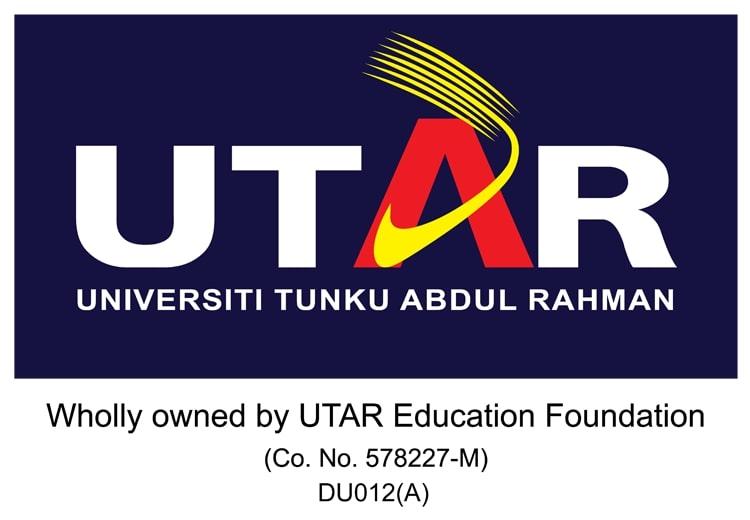
FINAL ASSESSMENT

**ANSWER SCRIPT**

**Candidate is required to fill in ALL the information below:**

|  |  |  |  |
| --- | --- | --- | --- |
| Name : (as stated in Student Identity Card) | TAN YING YAO | | |
| Faculty /Institute/ Centre: | LKC FES | Programme : | SOFTWARE ENGINEERING |
| Index No. (in numbers) : | U00339EBSEF | Index No. (in words) : | UZEROZEROTHREETHREENINEEBSEF |
| Course Code : | **UECS2354** | Course Description : | **SOFTWARE TESTING** |
| Submission Date : | 7/5/2020 | Time : | 11PM |

|  |  |  |
| --- | --- | --- |
| **QUESTION NUMBER** | **FOR EXAMINER’S USE ONLY** | |
| **MARKS** | |
| **Internal** | **External** |
| **1** |  |  |
| **2** |  |  |
| **3** |  |  |
| **4** |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| **TOTAL MARKS** |  |  |



**DECLARATION STATEMENT**

I, **TAN YING YAO** (Name), Student ID No. **1703648** hereby solemnly and fully declare and confirm that during my programme of study at Universiti Tunku Abdul Rahman, I shall abide and comply with all the rules, regulations and lawful instructions of Universiti Tunku Abdul Rahman and endeavour at all times to uphold the good name of the University.

I hereby declare that my submission for this Final Assessment is based on my original work, not plagiarised from any source(s) except for citations and quotations which have been duly acknowledged. I am fully aware that students who are suspected of violating this pledge are liable to be referred to the Examination Disciplinary Committee of the University.

|  |  |
| --- | --- |
| Programme: | SOFTWARE ENGINEERING |
| (Digital) Signature: | TAN YING YAO |
| Student’s I.C / Passport No.: | 971210-14-5673 |
| Index No: | U00339EBSEF |
| Date of Submission: | 7/5/2020 |

Index Number (in figure): U00339EBSEF Course Code**: UECS2354** Page: 1

1. A) The root cause of the issue is the lack of understanding regarding the importance of a software testing team. The defect is the system’s failure in calculating according to the rules of the new tax system. The effect is the loss of confidence of the system by the customer and reputation of the company. There are many corrective actions that can be taken such as sending the developer to attend software testing training. This can ensure that the developer obtains proper testing skills and attributes needed to test the software properly. The system should have also underwent beta testing and prototyping before pushing it to live environment immediately. The proper sequence testing can allow the tester to identify critical defects and fix them before it goes live. Finally, the company should have hired a proper software testing team. This helps lighten the developer’s burden to have to test and develop the software concurrently. Having a tester team can remove the developer bias towards the software they create.

B) In my opinion, any proper software company should have a software testing team. A proper software testing team can find bugs and provide software developers with the crucial information that can help with fixing the bugs. Software testing can also help gain confidence about the quality level of the system through acceptance testing. It can also prevent defects through early reviewing of the software. Most importantly, software testing can provide many information and aspects of the system and help management understand the system quality. Thus a software tester is extremely important to a software development cycle.

1. **A)**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ***Condition*** | **1** | **2** | **3** | **4** | **5** | **6** |
| **Income less than RM3000** | **N** | **Y** | **N** | **Y** | **Y** | **N** |
| **CGPA > 3.0** | **Y** | **N** | **Y** | **N** | **Y** | **N** |
| **Borrowed PTPTN Loan** | **N** | **N** | **Y** | **Y** | **-** | **-** |
| ***Action*** |  |  |  |  |  |  |
| **Scholarship** | **Y** | **Y** | **N** | **N** | **Y** | **N** |
| **Financial Assistance** | **N** | **N** | **Y** | **Y** | **Y** | **N** |

**B)** Boundary Value Analysis (BVA) and Equivalence Partitioning (EP) are usually used together to provide more coverage to the testing. However, this is only effective for Black Box or specification-based testing techniques and when there is large number of values for testing. These techniques are not suited for system above which only results in specific outcomes. Thus, it is not necessary for this test case.

Index Number (in figure): U00339EBSEF Course Code**: UECS2354** Page: 2

**C)**

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case Number | Test Summary | Test Data | Expected Result |
| **1** | **If Income more than RM 3000 and CGPA more than 3.0 but did not borrow PTPTN Loan** | **Income: RM 4000**  **CGPA : 3.69**  **PTPN: NO** | **Scholarship Provided** |
| **2** | **If Income less than RM 3000 and CGPA less than 3.0 but did not borrow PTPTN Loan** | **Income: RM 2000**  **CGPA : 2.00**  **PTPN: NO** | **Scholarship Provided** |
| **3** | **If Income more than RM 3000 and CGPA more than 3.0 and borrows PTPTN Loan** | **Income: RM 4000**  **CGPA : 3.69**  **PTPN: YES** | **Financial Assistance Provided** |
| **4** | **If Income less than RM 3000 and CGPA less than 3.0 and borrows PTPTN Loan** | **Income: RM 2000**  **CGPA : 2.69**  **PTPN: YES** | **Financial Assistance Provided** |
| **5** | **If Income less than RM 3000 and CGPA more than 3.0** | **Income: RM 2000**  **CGPA : 3.69**  **PTPN: -** | **Scholarship & Financial Assistance Provided** |
| **6** | **If Income more than RM 3000 and CGPA less than 3.0** | **Income: RM 5000**  **CGPA : 2.50**  **PTPN: -** | **No Scholarship & Financial Assistance Provided** |

**D) if (income < 0 || CGPA < 0 || CGPA > 4.0)**

**throw new IllegalArgumentException();**

**E) public class TestFinancialAid {**

**@Test**

**@Parameters(method = "checkStatusParameter")**

**public void checkStatusTest(double income, double CGPA, boolean PTPTNLoan, boolean scholar, boolean assist)**

**{ FinancialAid aid = new FinancialAid();**

**aid.checkStatus(income, CGPA, PTPTNLoan);**

**assertEquals(aid.haveScholarship(),scholar);**

**assertEquals(aid.haveFinanceAssist(),assist); }**

**private Object[] checkStatusParameter()**

**{ return new Object[] {**

**new Object[] {4000, 3.69, false, true, false},**

**new Object[] {2000, 2.00, false, true, false},**

**new Object[] {4000, 3.69, true, false, true},**

**new Object[] {2000, 2.69, true, false, true},**

**new Object[] {2000, 3.69, false, true, true},**

**new Object[] {5000, 2.50, false, false, false},**

**};}**

**@Test(expected=IllegalArgumentException.class)**

**@Parameters(method="checkStatusInvalidParameter")**

**public void checkStatusInvalid(double income, double CGPA, boolean PTPTNLoan) {**

**FinancialAid AID = new FinancialAid();**

**AID.checkStatus(income, CGPA, PTPTNLoan);}**

**private Object[] checkStatusInvalidParameter() {**

**return new Object[] {**

**new Object[] {-1000, 3.0, true},**

**new Object[] {-2000, 3.5, false},**

**new Object[] {500, 4.1, true},**

**new Object[] {1500, 5.0, false},**

**new Object[] {1500, -1.5, true},**

**new Object[] {2500, -1.0, false},**

**};}}**

1. **A)**

**BVA – calculateBonus()**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Sales Amount** | **<- -1** | **0-1000** | **1001-2000** | **2001-3000** | **3001-5000** | **5001 ->** |
|  | **(-1)** | **(0, 1000)** | **(1001, 2000)** | **(2001, 3000)** | **(3001, 5000)** | **(5001)** |
|  | **(Invalid)** | **(Valid)** | **(Valid)** | **(Valid)** | **(Valid)** | **(Valid)** |
| **Salary** | **INV** | **1500** | **1500** | **1500** | **1500** | **1500** |
| **Bonus** | **Salary + (Sales Amount \* 0.10)** | **Salary + (Sales Amount \* 0.15)** | **Salary + (Sales Amount \* 0.20)** | **Salary + (Sales Amount \* 0.25)** | **Salary + (Sales Amount \* 0.30)** |
| **Expected Results** | **(1500, 1600)** | **(1650.15, 1800)** | **(1900.2, 2100)** | **(2250.25, 2750)** | **(3000.3)** |

**EP-calculateBonus()**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Sales Amount** | **<- -1** | **0-1000** | **1001-2000** | **2001-3000** | **3001-5000** | **5001 ->** |
|  | **(-1)** | **(999)** | **(1999)** | **(2999)** | **(4999)** | **(5001)** |
| **Salary** | **INV** | **1500** | **1500** | **1500** | **1500** | **1500** |
| **Bonus** | **Salary + (Sales Amount \* 0.10)** | **Salary + (Sales Amount \* 0.15)** | **Salary + (Sales Amount \* 0.20)** | **Salary + (Sales Amount \* 0.25)** | **Salary + (Sales Amount \* 0.30)** |
| **Expected Results** | **1599.9** | **1799.85** | **2099.8** | **2749.75** | **3000.3** |

**B) if(salary < 0 || salesAmount < 0)**

**throw new IllegalArgumentException();**

**C) @Test**

**@Parameters(method = "calculateBonusParameter")**

**public void calculateBonusTest(double salary, double salesAmount, double bonus)**

**{**

**EmployeeIncome income = new EmployeeIncome();**

**double actualBonus = income.calculateBonus(salary, salesAmount);**

**assertEquals(actualBonus, bonus, 0.001);}**

**private Object[] calculateBonusParameter()**

**{ return new Object[] {**

**//assume salary is RM 1500**

**new Object[] {1500, 0, 1500},**

**new Object[] {1500, 1000, 1600},**

**new Object[] {1500, 1001, 1650.15},**

**new Object[] {1500, 2000, 1800},**

**new Object[] {1500, 2001, 1900.2},**

**new Object[] {1500, 3000, 2100},**

**new Object[] {1500, 3001, 2250.25},**

**new Object[] {1500, 5000, 2750},**

**new Object[] {1500, 5001, 3000.3},};}**

**@Test(expected=IllegalArgumentException.class)**

**@Parameters(method="calculateBonusInvalidParameter")**

**public void calculateBonusInvalid(double salary, double salesAmount) {**

**EmployeeIncome INC = new EmployeeIncome();**

**INC.calculateBonus(salary, salesAmount);}**

**private Object[] calculateBonusInvalidParameter() {**

**return new Object[] {**

**new Object[] {1500, -1},};}**

**D)**  **//public constructors used for testing purposes**

**public EmployeeIncome() {**

**emp = new Employee();}**

**public EmployeeIncome(Employee emp) {**

**this.emp = emp;}**

**E)** **// Methods to test getRevisedSalary()**

**@Test**

**@Parameters(method ="getRevisedSalaryParameter")**

**public void getRevisedSalaryTest(String name, String position, double RevisedSalary)**

**{ Employee employeeMock = mock(Employee.class);**

**when (employeeMock.EmployeeDetails(anyString())).thenReturn(position);**

**EmployeeIncome air = new EmployeeIncome(employeeMock);**

**air.calculateIncome(name);**

**double actualSalary= air.getRevisedSalary(1500.0);**

**assertEquals(actualSalary,RevisedSalary, 0.001);}**

**private Object[] getRevisedSalaryParameter() {**

**return new Object[] {**

**new Object[] {"Bob Tucker", "Junior executive", 1800.0},**

**new Object[] {"Bobby Tucker", "Senior executive", 2000.0},**

**new Object[] {"Billy Tucker", "Manager", 2200.0},};}**

**F)**

**// Methods to test updatePosition()**

**@Test(expected=IllegalArgumentException.class)**

**@Parameters(method="updatePositionInvalidParameter")**

**public void updatePositionTestInvalid(double salesAmount) {**

**Employee employeeMock = mock(Employee.class);**

**when (employeeMock.getSalesAmount()).thenReturn(salesAmount);**

**EmployeeIncome ats = new EmployeeIncome(employeeMock);**

**ats.calculateIncome(anyString());**

**ats.updatePosition(); }**

**private Object[] updatePositionInvalidParameter() {**

**return new Object[] {**

**new Object[] {-1.0}};}**

**@Test**

**@Parameters(method ="updatePositionParameter")**

**public void updatePositionTest(String position,double salesAmount,String expectedPosition)**

**{ Employee employeeMock = mock(Employee.class);**

**when (employeeMock.EmployeeDetails(anyString())).thenReturn(position);**

**when (employeeMock.getSalesAmount()).thenReturn(salesAmount);**

**EmployeeIncome ats = new EmployeeIncome(employeeMock);**

**ats.calculateIncome(anyString());**

**ats.updatePosition();**

**verify(employeeMock).updatePosition(expectedPosition);}**

**private Object[] updatePositionParameter(){**

**return new Object[] {**

**new Object[] {"Junior executive",0,"Junior executive"},**

**new Object[] {"Junior executive",999,"Junior executive"},**

**new Object[] {"Junior executive",1000,"Senior executive"},**

**new Object[] {"Junior executive",4999,"Senior executive"},**

**new Object[] {"Senior executive",5001,"Manager"}};}**

1. **A)**

|  |  |
| --- | --- |
| Bug Report 1: | **caseNO\_01** |
| Summary: | **Assertion Error; Expected Result is 4.5 but Actual Result is 2.5.** |
| Test Data: | **Distance: 1KM**  **Weight: 100G** |
| Expected Result: | **4.5** |
| Actual Result: | **2.5** |
| Impact: | **Critical** |
| Description of corrective action: | **Inform system developer to change the condition of getDeliveryCharge function. Distance should be greater or equal to 1 rather than greater than 1.** |

|  |  |
| --- | --- |
| Bug Report 2: | **caseNO\_02** |
| Summary: | **Assertion Error; Expected Result is 8.5 but Actual Result is 9.0.** |
| Test Data: | **Distance: 50KM**  **Weight: 1500G** |
| Expected Result: | **8.5** |
| Actual Result: | **9.0** |
| Impact: | **Critical** |
| Description of corrective action: | **Inform system developer to change the charge of getDeliveryCharge function. The value in the function is 8.5 instead of 9.0. This can be changed easily.** |

|  |  |
| --- | --- |
| Bug Report 3: | **caseNO\_03** |
| Summary: | **101KM is over the maximum distance allowed by the system. There is no condition that limits the distance allowed.** |
| Test Data: | **Distance: 101KM**  **Weight: 6000G** |
| Expected Result: | **15.0** |
| Actual Result: | **15.0** |
| Impact: | **Critical** |
| Description of corrective action: | **Inform system developer to change the condition of getDeliveryCharge function. Distance should have a maximum limit of 100KM only.** |

|  |  |
| --- | --- |
| Bug Report 4: | **caseNO\_04** |
| Summary: | **IllegalArgumentException; NULL is allowed as a test data input. It should not be allowed in the system.** |
| Test Data: | **Distance: NULL**  **Weight: 6000G** |
| Expected Result: | **15.0** |
| Actual Result: | **IllegalArgumentException** |
| Impact: | **Critical** |
| Description of corrective action: | **Inform system developer to change the getDeliveryCharge function. NULL should not be allowed to pass through the function.** |

|  |  |
| --- | --- |
| Bug Report 5: | **caseNO\_05** |
| Summary: | **Unexpected exception. This is due to the lack of condition checking where there are illegal exceptions.** |
| Test Data: | **Distance: 0KM**  **Weight: 1G**  **Distance: 101KM**  **Weight: 1G**  **Distance: 1KM**  **Weight: 0G**  **Distance: 1KM**  **Weight: 10000G** |
| Expected Result: | **2.5** |
| Actual Result: | **Unexpected exception.** |
| Impact: | **Critical** |
| Description of corrective action: | **Inform system developer to change the getDeliveryCharge function. They should throw new IllegalArgumentException that meets the requirements of the system.** |

**B)** The incident report should be submitted to an investigation team to further study and look for deeper causes. An investigation should be conducted by those who are competent in collecting and analyzing information and evidence gathered from the incident report. Those conducting the investigation should be knowledgeable in the software development. These issue can then be fix through a bug report once it is established.

**C)** In my opinion, the incident report given was unprofessional and too short to provide relevant information. It failed to include several key information such as configuration of system, date of report, life cycle process, change history and the degree of impact on stakeholders interest. It is also unprofessional as threatening the tester does not help with the issue and only further strain the relationship between the tester and developer. The incident report should proper describe a test event that needs further investigation, especially a bug and should include test incident report identifier, summary, description and impact of failure.

D) public double getDeliveryCharge(int distance, int weight){

//throw illegal exceptions

if (distance <= 0 || distance >100 || weight <= 0 || weight >=10000)

throw new IllegalArgumentException();

//fixes caseNO01 by adding proper condition

if(distance >= 1 && distance < 30){

if(weight < 1000)

charges = 2.5;

else if(weight < 5000)

charges = 5.0;

else if(weight < 10000)

charges = 7.5;}

else{

if(weight < 1000)

charges = 4.5;

else if(weight < 5000)

//fixes caseNO02 by changing to proper value.

//charges = 8.5;

charges = 9.0;

else if(weight < 10000)

charges = 15.0;}

return charges;}

public class TestDeliveryCharges {

@Test

@Parameters(method="getParam")

public void testGetChargeRate(int distance, int weight, double expected){

DeliveryCharges DC = new DeliveryCharges();

double actual = DC.getDeliveryCharge(distance, weight);

assertEquals(actual, expected, 0);}

private Object[] getParam(){

return new Object[]{

//error 1

new Object[] {1,100,2.5},

new Object[] {25,1500,5.0},

new Object[] {14,6578,7.5},

//error 2

new Object[] {50,1500,9.0},

//error 3 -101 is out of the maximum

//new Object[] {101,6000,15.0},

//error 4 -null should not be allowed

//new Object[] {null,6000,15.0},

new Object[] {44,9999,15.00}

};

}

@Test(expected=IllegalArgumentException.class)

@Parameters(method="getParamIllegal")

public void testInvalidValues(int distance, int weight, double expected){

DeliveryCharges DC = new DeliveryCharges();

assertEquals(DC.getDeliveryCharge(distance, weight), expected, 0);

}

private Object[] getParamIllegal(){

return new Object[]{

//error 5

new Object[] {0, 1, 2.5},

new Object[] {101, 1, 2.5},

new Object[] {1, 0, 2.5},

new Object[] {1, 10000, 2.5},

//moved to test illegal parameters

new Object[] {101,6000,15.0},

new Object[] {null,6000,15.0},

};

}

}

**Application Code:**

**DeliveryCharge.Java**

package FA;

public class DeliveryCharges {

int weight=0;

int distance=0;

double charges=0;

public double getDeliveryCharge(int distance, int weight){

//throw illegal exceptions

if (distance <= 0 || distance >100 || weight <= 0 || weight >=10000)

throw new IllegalArgumentException();

//fixes caseNO01 by adding proper condition

if(distance >= 1 && distance < 30){

if(weight < 1000)

charges = 2.5;

else if(weight < 5000)

charges = 5.0;

else if(weight < 10000)

charges = 7.5;

}

else{

if(weight < 1000)

charges = 4.5;

else if(weight < 5000)

//fixes caseNO02 by changing to proper value.

//charges = 8.5;

charges = 9.0;

else if(weight < 10000)

charges = 15.0;

}

return charges;

}

} **Employee.Java**

package FA;

class Employee {

String name, position;

double salary;

double salesAmount;

int staffId;

public String EmployeeDetails(String name) {

/\*\*

\* codes check and return employee's position based on name

\* there are 3 category:

\* I) Junior executive

\* II) Senior executive

\* III) Manager

\*/

return position;

}

public void calculateIncome() {

// codes to calculate income

// the sales amount will be stored in variable salesAmount

}

public double getSalesAmount() {

return salesAmount;

}

public String updatePosition(String position) {

// update employee position based on the sales amount

return position;

}

public double getSalary(double salary) {

//employee salary

return salary;

}

} **EmployeeIncome.Java**

}package FA;

public class EmployeeIncome {

String name="";

String position="";

double salary=0;

double bonus=0;

double salesAmount=0;

Employee emp;

//public constructors used for testing purposes

public EmployeeIncome() {

emp = new Employee();

}

public EmployeeIncome(Employee emp) {

this.emp = emp;

}

public void calculateIncome (String name) {

position = emp.EmployeeDetails(name);

emp.calculateIncome();

salesAmount = emp.getSalesAmount();

}

// calculate the revised salary based on the position

public double getRevisedSalary(double salary) {

double revisedSalary = 0.0;

if (position.equals("Junior executive")) {

revisedSalary = salary + 300;

}

else if (position.equals("Senior executive")) {

revisedSalary = salary + 500;

}

else {

revisedSalary = salary + 700;

}

return revisedSalary;

}

// update employee's position through employee's updatePosition() method

public void updatePosition() {

String newPosition = "";

if (salesAmount < 0)

throw new IllegalArgumentException();

if (position.equals("Junior executive")) {

if (salesAmount >= 1000)

newPosition = "Senior executive";

else

newPosition = "Junior executive";

}

else if (position.equals("Senior executive")) {

if (salesAmount >= 5000)

newPosition = "Manager";

else if (salesAmount >= 1000 && salesAmount < 5000)

newPosition = "Senior executive";

else

newPosition = "Junior executive";

}

else {

if (salesAmount >= 5000)

newPosition = "Manager";

else

newPosition = "Senior executive";

}

emp.updatePosition(newPosition);

}

public double calculateBonus(double salary, double salesAmount) {

//calculate bonus based on sales amount

if(salary < 0 || salesAmount < 0)

throw new IllegalArgumentException();

if(salesAmount<=1000) {

bonus = salary + (salesAmount \* 0.10);

}

else if (salesAmount > 1000 && salesAmount <= 2000) {

bonus = salary + (salesAmount \* 0.15);

}

else if (salesAmount > 2000 && salesAmount <= 3000) {

bonus = salary + (salesAmount \* 0.20);

}

else if (salesAmount > 3000 && salesAmount <= 5000) {

bonus = salary + (salesAmount \* 0.25);

}

else {

bonus = salary + (salesAmount \* 0.30);

}

return bonus;

}

**FinancialAid.Java**

}

package FA;

public class FinancialAid {

private boolean scholarship, financeAssist;

public FinancialAid() {

scholarship = financeAssist = false;

}

public boolean haveScholarship() {

return scholarship;

}

public boolean haveFinanceAssist() {

return financeAssist;

}

public void checkStatus(double income, double CGPA, boolean PTPTNLoan) {

if (income < 0 || CGPA < 0 || CGPA > 4.0)

throw new IllegalArgumentException();

if (income < 3000.0) {

if (CGPA > 3.0) {

scholarship = financeAssist = true;

} else {

if (PTPTNLoan)

financeAssist = true;

else

scholarship = true;

}

}

else {

if (CGPA > 3.0) {

if (PTPTNLoan)

financeAssist = true;

else

scholarship = true;

}

}

**Test Code:**

**TestDeliveryCharge.Java**

}

package FA;

import junitparams.JUnitParamsRunner;

import junitparams.Parameters;

import org.junit.Test;

import org.junit.runner.RunWith;

import static org.junit.Assert.assertEquals;

@RunWith(JUnitParamsRunner.class)

public class TestDeliveryCharges {

@Test

@Parameters(method="getParam")

public void testGetChargeRate(int distance, int weight, double expected){

DeliveryCharges DC = new DeliveryCharges();

double actual = DC.getDeliveryCharge(distance, weight);

assertEquals(actual, expected, 0);

}

private Object[] getParam(){

return new Object[]{

//error 1

new Object[] {1,100,2.5},

new Object[] {25,1500,5.0},

new Object[] {14,6578,7.5},

//error 2

new Object[] {50,1500,9.0},

//error 3 -101 is out of the maximum

//new Object[] {101,6000,15.0},

//error 4 -null should not be allowed

//new Object[] {null,6000,15.0},

new Object[] {44,9999,15.00}

};

}

@Test(expected=IllegalArgumentException.class)

@Parameters(method="getParamIllegal")

public void testInvalidValues(int distance, int weight, double expected){

DeliveryCharges DC = new DeliveryCharges();

assertEquals(DC.getDeliveryCharge(distance, weight), expected, 0);

}

private Object[] getParamIllegal(){

return new Object[]{

//error 5

new Object[] {0, 1, 2.5},

new Object[] {101, 1, 2.5},

new Object[] {1, 0, 2.5},

new Object[] {1, 10000, 2.5},

//moved to test illegal parameters

new Object[] {101,6000,15.0},

new Object[] {null,6000,15.0},

};

}

}

}

**TestEmployeeIncome.Java**

package FA;

import static org.junit.Assert.assertEquals;

import static org.mockito.Matchers.anyString;

import static org.mockito.Mockito.\*;

import org.junit.Test;

import org.junit.runner.RunWith;

import junitparams.JUnitParamsRunner;

import junitparams.Parameters;

@RunWith(JUnitParamsRunner.class)

public class TestEmployeeIncome {

// Methods to test calculateBonus()

@Test

@Parameters(method = "calculateBonusParameter")

public void calculateBonusTest(double salary, double salesAmount, double bonus)

{

EmployeeIncome income = new EmployeeIncome();

double actualBonus = income.calculateBonus(salary, salesAmount);

assertEquals(actualBonus, bonus, 0.001);

}

private Object[] calculateBonusParameter()

{

return new Object[] {

//assume salary is RM 1500

new Object[] {1500, 0, 1500},

new Object[] {1500, 1000, 1600},

new Object[] {1500, 1001, 1650.15},

new Object[] {1500, 2000, 1800},

new Object[] {1500, 2001, 1900.2},

new Object[] {1500, 3000, 2100},

new Object[] {1500, 3001, 2250.25},

new Object[] {1500, 5000, 2750},

new Object[] {1500, 5001, 3000.3},

};

}

@Test(expected=IllegalArgumentException.class)

@Parameters(method="calculateBonusInvalidParameter")

public void calculateBonusInvalid(double salary, double salesAmount) {

EmployeeIncome INC = new EmployeeIncome();

INC.calculateBonus(salary, salesAmount);

}

private Object[] calculateBonusInvalidParameter() {

return new Object[] {

new Object[] {1500, -1},

};

}

// Methods to test getRevisedSalary()

@Test

@Parameters(method ="getRevisedSalaryParameter")

public void getRevisedSalaryTest(String name, String position, double RevisedSalary)

{

Employee employeeMock = mock(Employee.class);

when (employeeMock.EmployeeDetails(anyString())).thenReturn(position);

EmployeeIncome air = new EmployeeIncome(employeeMock);

air.calculateIncome(name);

double actualSalary= air.getRevisedSalary(1500.0);

assertEquals(actualSalary,RevisedSalary, 0.001);

}

private Object[] getRevisedSalaryParameter() {

return new Object[] {

new Object[] {"Bob Tucker", "Junior executive", 1800.0},

new Object[] {"Bobby Tucker", "Senior executive", 2000.0},

new Object[] {"Billy Tucker", "Manager", 2200.0},

};

}

// Methods to test updatePosition()

@Test(expected=IllegalArgumentException.class)

@Parameters(method="updatePositionInvalidParameter")

public void updatePositionTestInvalid(double salesAmount) {

Employee employeeMock = mock(Employee.class);

when (employeeMock.getSalesAmount()).thenReturn(salesAmount);

EmployeeIncome ats = new EmployeeIncome(employeeMock);

ats.calculateIncome(anyString());

ats.updatePosition();

}

private Object[] updatePositionInvalidParameter() {

return new Object[] {

new Object[] {-1.0}

};

}

@Test

@Parameters(method ="updatePositionParameter")

public void updatePositionTest(String position,double salesAmount,String expectedPosition)

{

Employee employeeMock = mock(Employee.class);

when (employeeMock.EmployeeDetails(anyString())).thenReturn(position);

when (employeeMock.getSalesAmount()).thenReturn(salesAmount);

EmployeeIncome ats = new EmployeeIncome(employeeMock);

ats.calculateIncome(anyString());

ats.updatePosition();

verify(employeeMock).updatePosition(expectedPosition);

}

private Object[] updatePositionParameter(){

return new Object[] {

new Object[] {"Junior executive",0,"Junior executive"},

new Object[] {"Junior executive",999,"Junior executive"},

new Object[] {"Junior executive",1000,"Senior executive"},

new Object[] {"Junior executive",4999,"Senior executive"},

new Object[] {"Senior executive",5001,"Manager"}

};

}

}

**TestFinancialAid.Java**

package FA;

import junitparams.JUnitParamsRunner;

import junitparams.Parameters;

import org.junit.Test;

import org.junit.runner.RunWith;

import static org.junit.Assert.assertEquals;

@RunWith(JUnitParamsRunner.class)

public class TestFinancialAid {

@Test

@Parameters(method = "checkStatusParameter")

public void checkStatusTest(double income, double CGPA, boolean PTPTNLoan, boolean scholar, boolean assist)

{

FinancialAid aid = new FinancialAid();

aid.checkStatus(income, CGPA, PTPTNLoan);

assertEquals(aid.haveScholarship(),scholar);

assertEquals(aid.haveFinanceAssist(),assist);

}

private Object[] checkStatusParameter()

{

return new Object[] {

new Object[] {4000, 3.69, false, true, false},

new Object[] {2000, 2.00, false, true, false},

new Object[] {4000, 3.69, true, false, true},

new Object[] {2000, 2.69, true, false, true},

new Object[] {2000, 3.69, false, true, true},

new Object[] {5000, 2.50, false, false, false},

};

}

@Test(expected=IllegalArgumentException.class)

@Parameters(method="checkStatusInvalidParameter")

public void checkStatusInvalid(double income, double CGPA, boolean PTPTNLoan) {

FinancialAid AID = new FinancialAid();

AID.checkStatus(income, CGPA, PTPTNLoan);

}

private Object[] checkStatusInvalidParameter() {

return new Object[] {

new Object[] {-1000, 3.0, true},

new Object[] {-2000, 3.5, false},

new Object[] {500, 4.1, true},

new Object[] {1500, 5.0, false},

new Object[] {1500, -1.5, true},

new Object[] {2500, -1.0, false},

};

}

}