|  |  |
| --- | --- |
| **Your name** |  |
| **Your login id** |  |

**Part A – Logbook:**

**Logbook 1 – Inheritance**

**Basic Information**

|  |  |
| --- | --- |
| 1.1 Student name | **Nkem Akwari** |
| 1.2 Who did you work with? Name and/or id | **Trevor Kiggundu** |
| 1.3 Which lab topic does this document relate to? | 1. Inheritance |
| 1.4 How well do you feel you have done? | * I have completed the exercise and am totally satisfied with my work |
| 1.5 Briefly explain your answer to question 1.4 | **I feel as I have completed the work to the specification detailed in the lab document and this is reflected with how my code looks and the end result’s functionality in terms of using inheritance to streamline a system and cut down on code.** |

1. **Implementation**

2.1 Annotated screen shots demonstrating what you have achieved.

Figure 1- basic hospital program running without amending, original button outputs those records, inheritance does nothing:

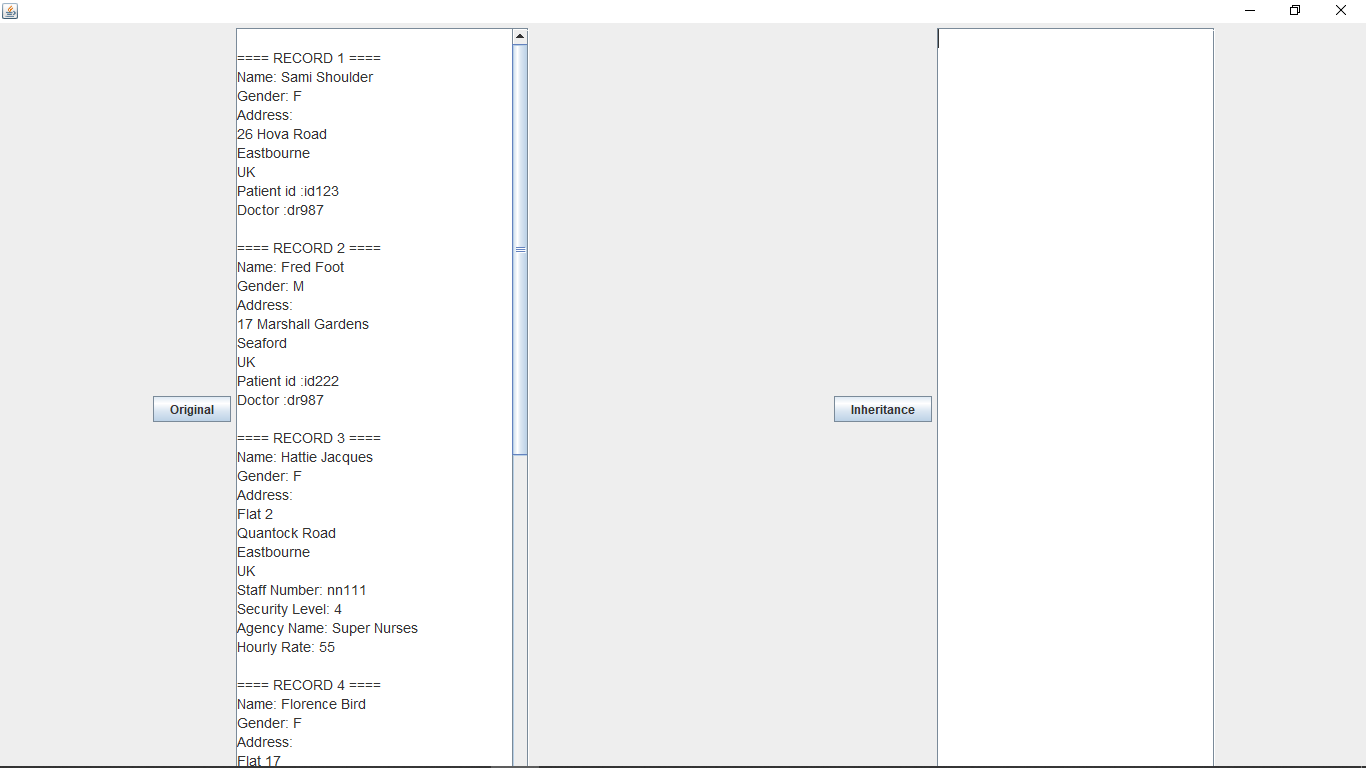
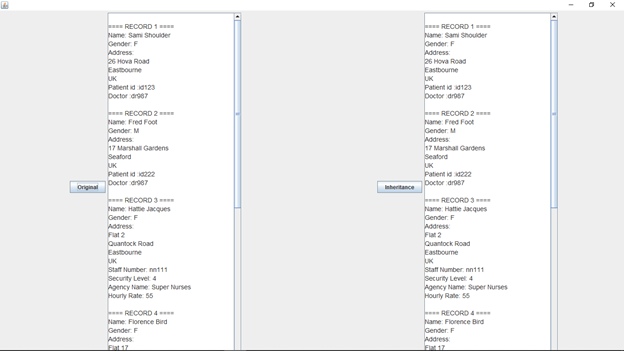


Figure 2 - Added correct functionality to handleinheritancebutton(). Output is the same as original:



2.2 Copy and paste **code that you wrote or amended**. Please **format** it nicely and **make it easy** for the tutor to see and read your code.

**//Creation of person class to hold basic fields**

public class Person {

protected String firstName;

protected String familyName;

protected char gender;

protected PostalAddress postalAddress;

public Person(String firstName, String familyName, char gender, PostalAddress postalAddress) {

this.firstName = firstName;

this.familyName = familyName;

this.gender = gender;

this.postalAddress = postalAddress;

}

public Person() {

this.firstName = "";

this.familyName = "";

this.gender = 'U';

this.postalAddress = new PostalAddress();

}

public String getFirstName() {

return firstName;

}

public String getFamilyName() {

return familyName;

}

public char getGender() {

return gender;

}

public PostalAddress getPostalAddress() {

return postalAddress;

}

}

**//changed aspects of AgencyStaff to support inheritance of person**

public class AgencyStaff extends Person {

private String staffNumber;

private int securityLevel;

private String agencyName;

private int hourlyRate;

public AgencyStaff(String agencyName, int hourlyRate, String staffNumber, int securityLevel, String firstName, String familyName, char gender, PostalAddress postalAddress) {

super(firstName, familyName, gender, postalAddress);

this.staffNumber = staffNumber;

this.securityLevel = securityLevel;

this.agencyName = agencyName;

this.hourlyRate = hourlyRate;

}

public AgencyStaff() {

this.firstName = super.firstName ;

this.familyName = super.familyName;

this.gender = super.gender;

this.postalAddress = super.postalAddress;

this.staffNumber = "";

this.agencyName = "";

}

public String getStaffNumber() {

return staffNumber;

}

public int getSecurityLevel() {

return securityLevel;

}

public String getAgencyName() {

return agencyName;

}

public int getHourlyRate() {

return hourlyRate;

}

}

**//changed aspects of directemployee to support inheritance of person**

public class DirectEmployee extends Person {

private String staffNumber;

private int securityLevel;

private String grade;

private int salary;

public DirectEmployee(String grade, int salary, String staffNumber, int securityLevel, String firstName, String familyName, char gender, PostalAddress postalAddress) {

super(firstName, familyName, gender, postalAddress);

this.staffNumber = staffNumber;

this.securityLevel = securityLevel;

this.grade = grade;

this.salary = salary;

}

public DirectEmployee() {

this.firstName = super.firstName ;

this.familyName = super.familyName;

this.gender = super.gender;

this.postalAddress = super.postalAddress;

this.staffNumber = "";

this.grade = "";

}

public String getStaffNumber() {

return staffNumber;

}

public int getSecurityLevel() {

return securityLevel;

}

public String getGrade() {

return grade;

}

public int getSalary() {

return salary;

}

}

**//Changed aspects of Patient to support inheritance of person**

public class Patient extends Person{

private String patientId;

private String doctorStaffNumber;

public Patient(String patientId, String doctorStaffNumber, String firstName, String familyName, char gender, PostalAddress postalAddress) {

super(firstName, familyName, gender, postalAddress);

this.patientId = patientId;

this.doctorStaffNumber = doctorStaffNumber;

}

public Patient() {

this.firstName = super.firstName ;

this.familyName = super.familyName;

this.gender = super.gender;

this.postalAddress = super.postalAddress;

this.patientId = "";

this.doctorStaffNumber = "";

}

public String getPatientId() {

return patientId;

}

public String getDoctorStaffNumber() {

return doctorStaffNumber;

}

}

//created code for handleinheritancebutton() that shows how inheritance cuts down code

private void handleInheritanceButton() {

// set up the data

List<Object> listOfRecords = new ArrayList<>();

listOfRecords.add(patient1);

listOfRecords.add(patient2);

listOfRecords.add(agencystaff1);

listOfRecords.add(agencystaff2);

listOfRecords.add(directEmployee1);

listOfRecords.add(directEmployee2);

txtInheritance.setText(""); // clear the output

// Loop through the data displaying the details

int count = 0;

for (Object o : listOfRecords) {

count++;

/\*\*starts with the superclass, which all current objects are a part of.

\*Through polymorphism, this means each entry that goes through this method

\*will display the instance variables that it shares with person first

\*/

if (o instanceof Person) {

Person pe = (Person) o;

txtInheritance.append(String.format("\n==== RECORD %d ====\n", count));

txtInheritance.append("Name: " + pe.getFirstName() + " " + pe.getFamilyName() + "\n");

txtInheritance.append("Gender: " + pe.getGender() + "\n");

txtInheritance.append("Address:\n" + pe.getPostalAddress().getDisplayableAddress() + "\n");

}

/\*

\*With the rest, it will check if it fits into any of the

\*subclasses during each if loop, and if so, adds the rest of the

\*relevant instance variables to its record before going to the

\*next arraylistentry and doing the same until the arraylist has

\*been iterated through.

\*/

if (o instanceof AgencyStaff) {

AgencyStaff as = (AgencyStaff) o;

txtInheritance.append("Staff Number: " + as.getStaffNumber() + "\n");

txtInheritance.append("Security Level: " + as.getSecurityLevel() + "\n");

txtInheritance.append("Agency Name: " + as.getAgencyName() + "\n");

txtInheritance.append("Hourly Rate: " + as.getHourlyRate() + "\n");

} else if (o instanceof DirectEmployee) {

DirectEmployee de = (DirectEmployee) o;

txtInheritance.append("Staff Number: " + de.getStaffNumber() + "\n");

txtInheritance.append("Security Level: " + de.getSecurityLevel() + "\n");

txtInheritance.append("Grade :" + de.getGrade() + "\n");

txtInheritance.append(String.format("Salary: £%,d\n", de.getSalary()));

} else if (o instanceof Patient) {

Patient pa = (Patient) o;

txtInheritance.append("Patient id :" + pa.getPatientId() + "\n");

txtInheritance.append("Doctor :" + pa.getDoctorStaffNumber() + "\n");

}

}

// Loop through the data displaying the details

}

**Logbook 2 – JUnit testing**

**Basic Information**

|  |  |
| --- | --- |
| 1.1 Student name | **Trevor Kiggundu** |
| 1.2 Who did you work with? Name and/or id | **Nkem Akwari** |
| 1.3 Which lab topic does this document relate to? | JUnit |
| 1.4 How well do you feel you have done? | * I have completed the exercise and am totally satisfied with my work |
| 1.5 Briefly explain your answer to question 1.4 | **I feel as I have completed the work to the specification detailed in the lab document. This is justified by the tests and explanations being at the satisfactory level needed for completion.** |

1. **Implementation**

2.1 Annotated screen shots demonstrating what you have achieved.

Figure 1- Running the JUnit tests and noting which ones failed to pass the test:

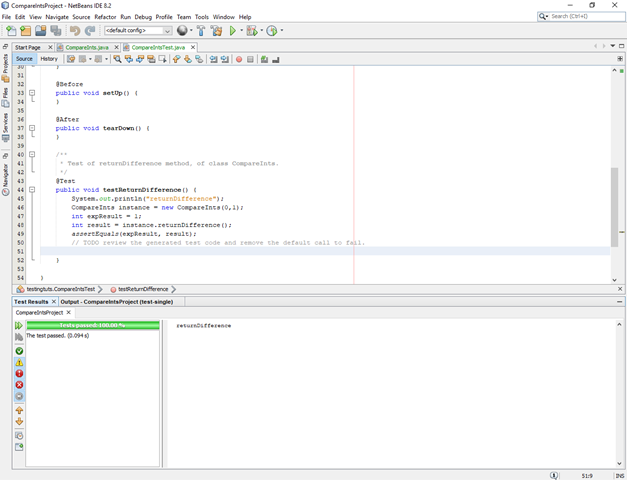
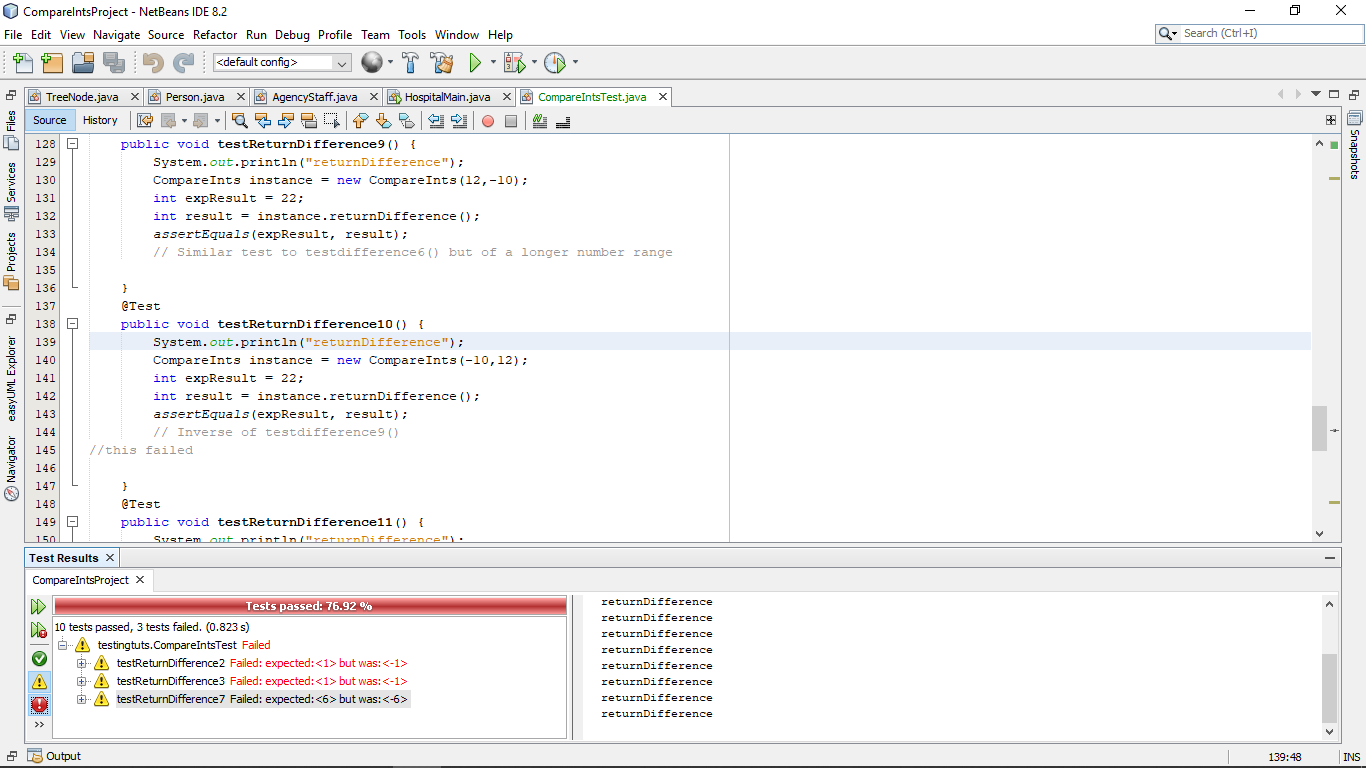


Figure 2 -Creation of the first testreturndifference(), testing, three of the 14 tests failed:



2.2 Copy and paste **code that you wrote or amended**. Please **format** it nicely and **make it easy** for the tutor to see and read your code.

**//Test package for compareints**

public class CompareIntsTest {

public CompareIntsTest() {

}

@BeforeClass

public static void setUpClass() {

}

@AfterClass

public static void tearDownClass() {

}

@Before

public void setUp() {

}

@After

public void tearDown() {

}

/\*\*

\* Test of returnDifference method, of class CompareInts.

\*/

@Test

public void testReturnDifference() {

System.out.println("returnDifference");

CompareInts instance = new CompareInts(0,1);

int expResult = 1;

int result = instance.returnDifference();

assertEquals(expResult, result);

// basic example test given by document

}

@Test

public void testReturnDifference2() {

System.out.println("returnDifference");

CompareInts instance = new CompareInts(-1,0);

int expResult = 1;

int result = instance.returnDifference();

assertEquals(expResult, result);

//Test used to test if a negative as "one" can work with "two" being 0

//This failed.

}

@Test

public void testReturnDifference3() {

System.out.println("returnDifference");

CompareInts instance = new CompareInts(0,-1);

int expResult = 1;

int result = instance.returnDifference();

assertEquals(expResult, result);

//Inverse of testreturndifference2(), negative in second place, postive in first

//this failed

}

@Test

public void testReturnDifference4() {

System.out.println("returnDifference");

CompareInts instance = new CompareInts(10,-1);

int expResult = 11;

int result = instance.returnDifference();

assertEquals(expResult, result);

//Testing if a positive as the first number can be taken away from a negative which is the second number

}

@Test

public void testReturnDifference5() {

System.out.println("returnDifference");

CompareInts instance = new CompareInts(2,2);

int expResult = 0;

int result = instance.returnDifference();

assertEquals(expResult, result);

//Testing if two of the same number can work with this class

}

@Test

public void testReturnDifference6() {

System.out.println("returnDifference");

CompareInts instance = new CompareInts(5,-5);

int expResult = 10;

int result = instance.returnDifference();

assertEquals(expResult, result);

// Testing if a positve value with the same digit value can be used with a negative of the same digit number

}

@Test

public void testReturnDifference7() {

System.out.println("returnDifference");

CompareInts instance = new CompareInts(-7,-1);

int expResult = 6;

int result = instance.returnDifference();

assertEquals(expResult, result);

//Another test testing if a negative as the first number will work correctly with a positive as the second

//this failed

}

@Test

public void testReturnDifference8() {

System.out.println("returnDifference");

CompareInts instance = new CompareInts(0,4);

int expResult = 4;

int result = instance.returnDifference();

assertEquals(expResult, result);

// Testing if zero as first number works with a positive value as second

}

@Test

public void testReturnDifference9() {

System.out.println("returnDifference");

CompareInts instance = new CompareInts(12,-10);

int expResult = 22;

int result = instance.returnDifference();

assertEquals(expResult, result);

// Similar test to testdifference6() but of a longer number range

}

@Test

public void testReturnDifference10() {

System.out.println("returnDifference");

CompareInts instance = new CompareInts(-10,12);

int expResult = 22;

int result = instance.returnDifference();

assertEquals(expResult, result);

// Inverse of testdifference9()

}

@Test

public void testReturnDifference11() {

System.out.println("returnDifference");

CompareInts instance = new CompareInts(-1,-1);

int expResult = 0;

int result = instance.returnDifference();

assertEquals(expResult, result);

//Test to see if two negative numbers work

}

@Test

public void testReturnDifference12() {

System.out.println("returnDifference");

CompareInts instance = new CompareInts(2,6);

int expResult = 4;

int result = instance.returnDifference();

assertEquals(expResult, result);

//basic test to see if these postive numbers work

}

@Test

public void testReturnDifference13() {

System.out.println("returnDifference");

CompareInts instance = new CompareInts(6,2);

int expResult = 4;

int result = instance.returnDifference();

assertEquals(expResult, result);

// inverse positions from testreturndifference12()

}

**Logbook 3 – Design Patterns**

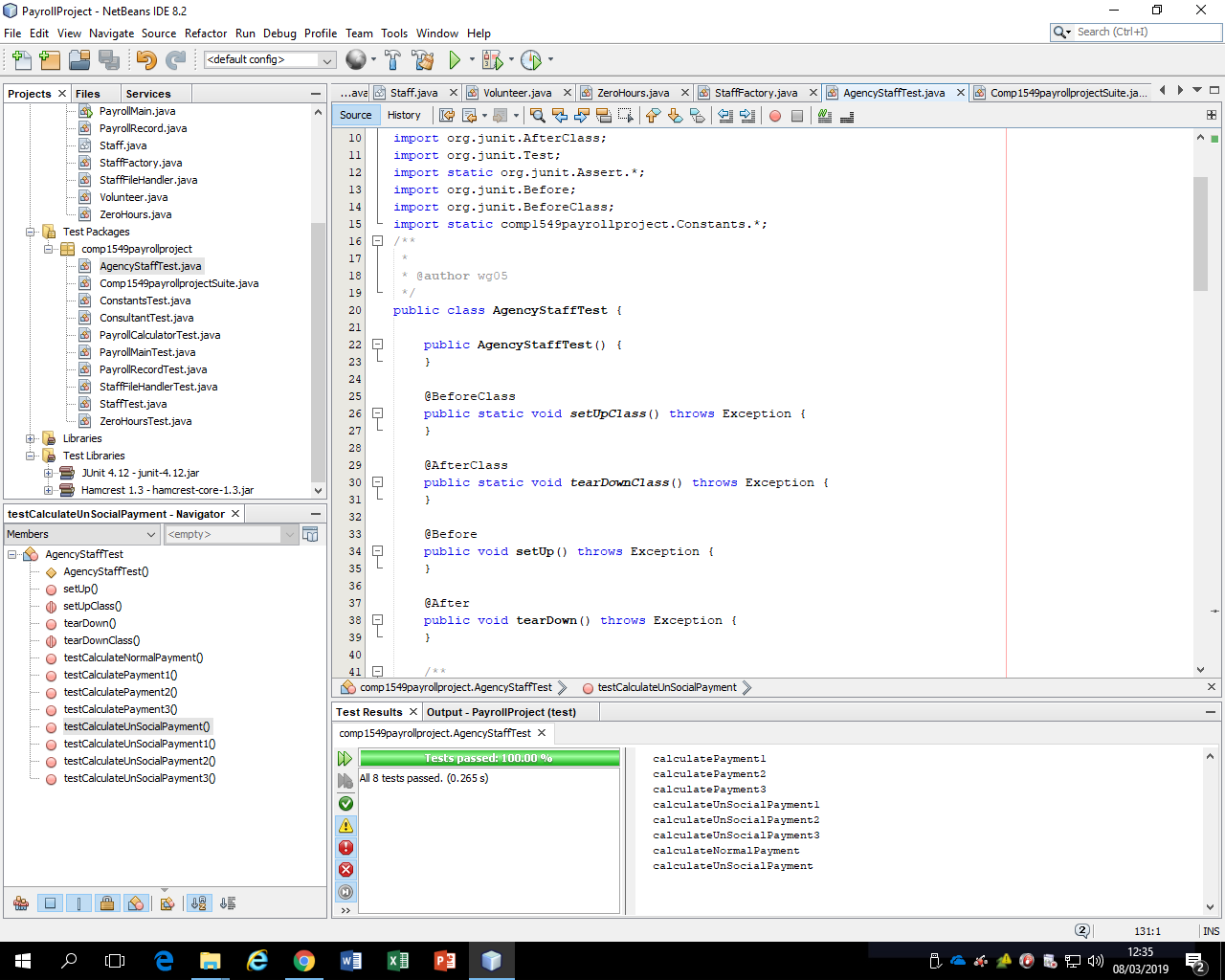
**Basic Information**

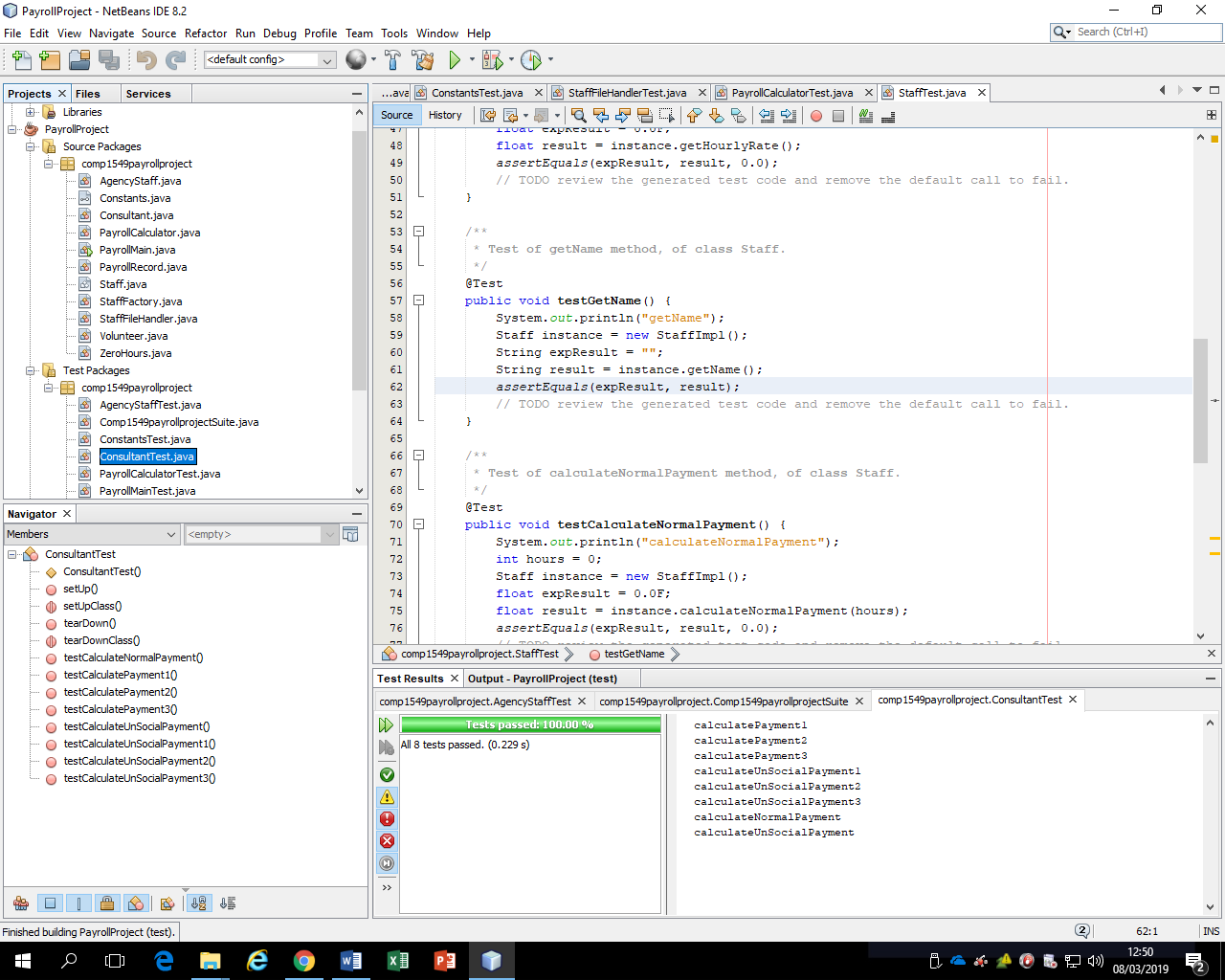
|  |  |
| --- | --- |
| 1.1 Student name | **Nkem Akwari** |
| 1.2 Who did you work with? Name and/or id | **Trevor** |
| 1.3 Which lab topic does this document relate to? | Design Patterns |
| 1.4 How well do you feel you have done? | * I have completed the exercise and am totally satisfied with my work |
| 1.5 Briefly explain your answer to question 1.4 | **I feel as I have completed the work to the specification detailed in the lab document and this is reflected with how my code looks and the end result’s functionality in terms of applying the proper design patterns to the sample code, and adequately testing if this works correctly through JUnit** |

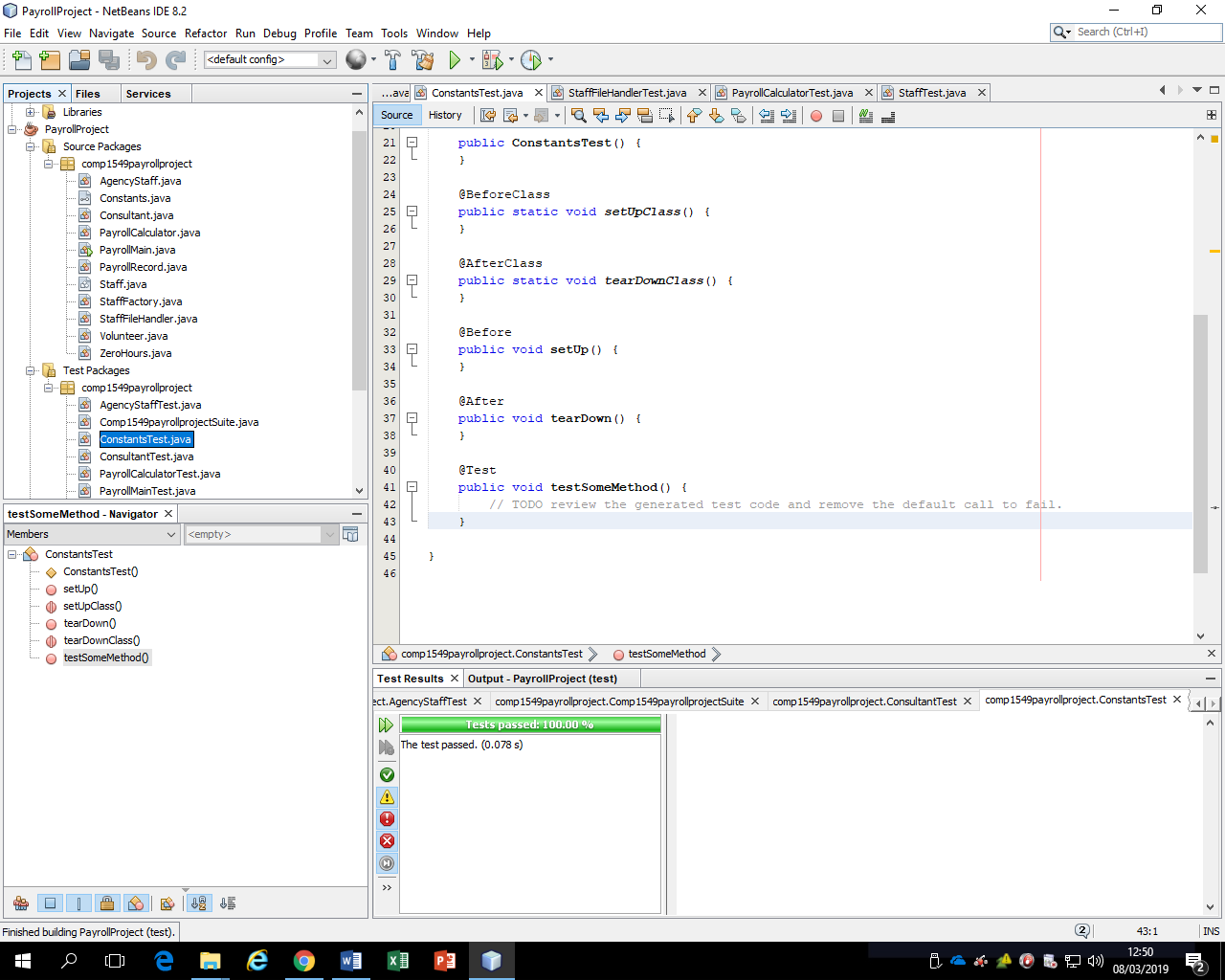
1. **Implementation**

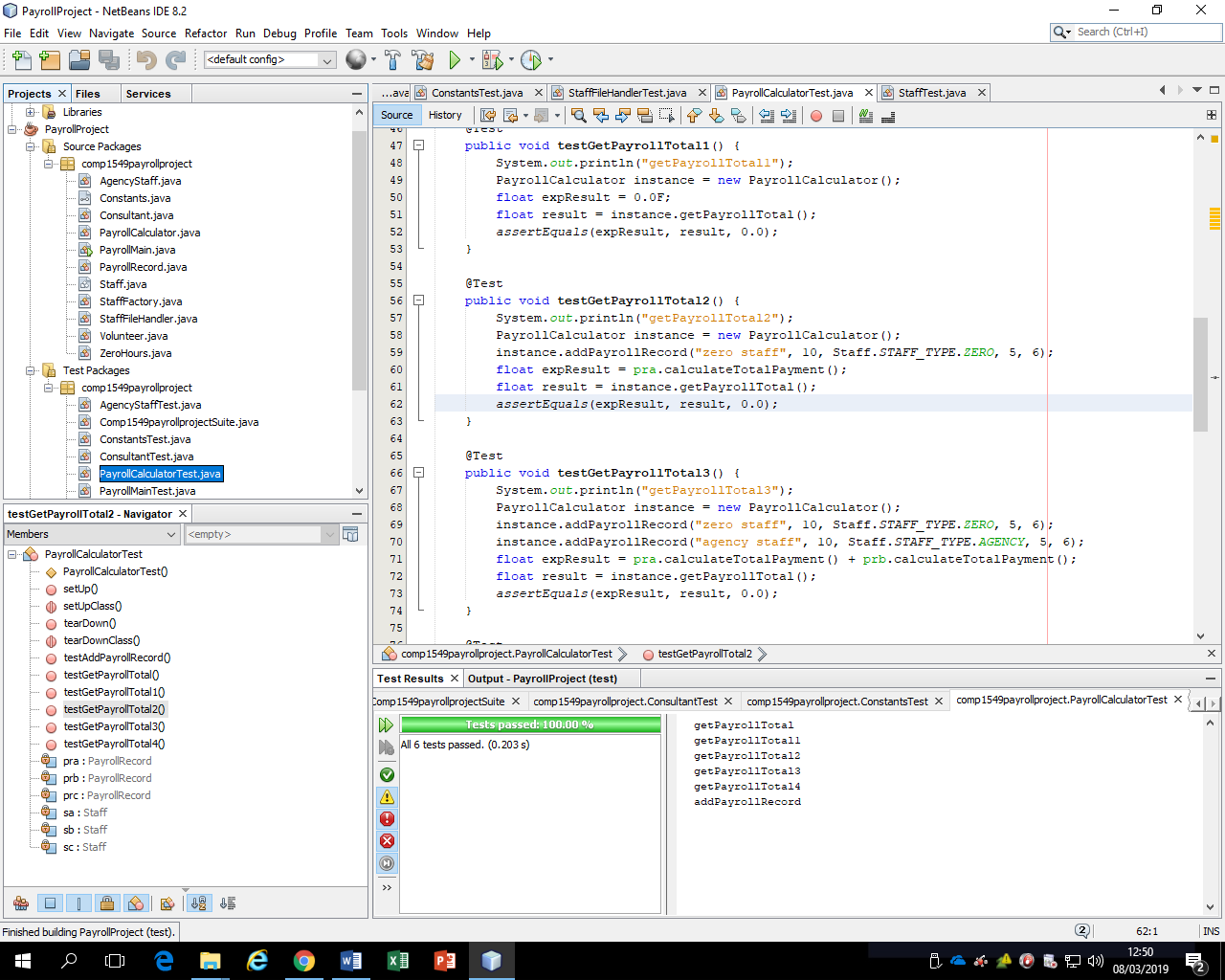
2.1 Annotated screen shots demonstrating what you have achieved.

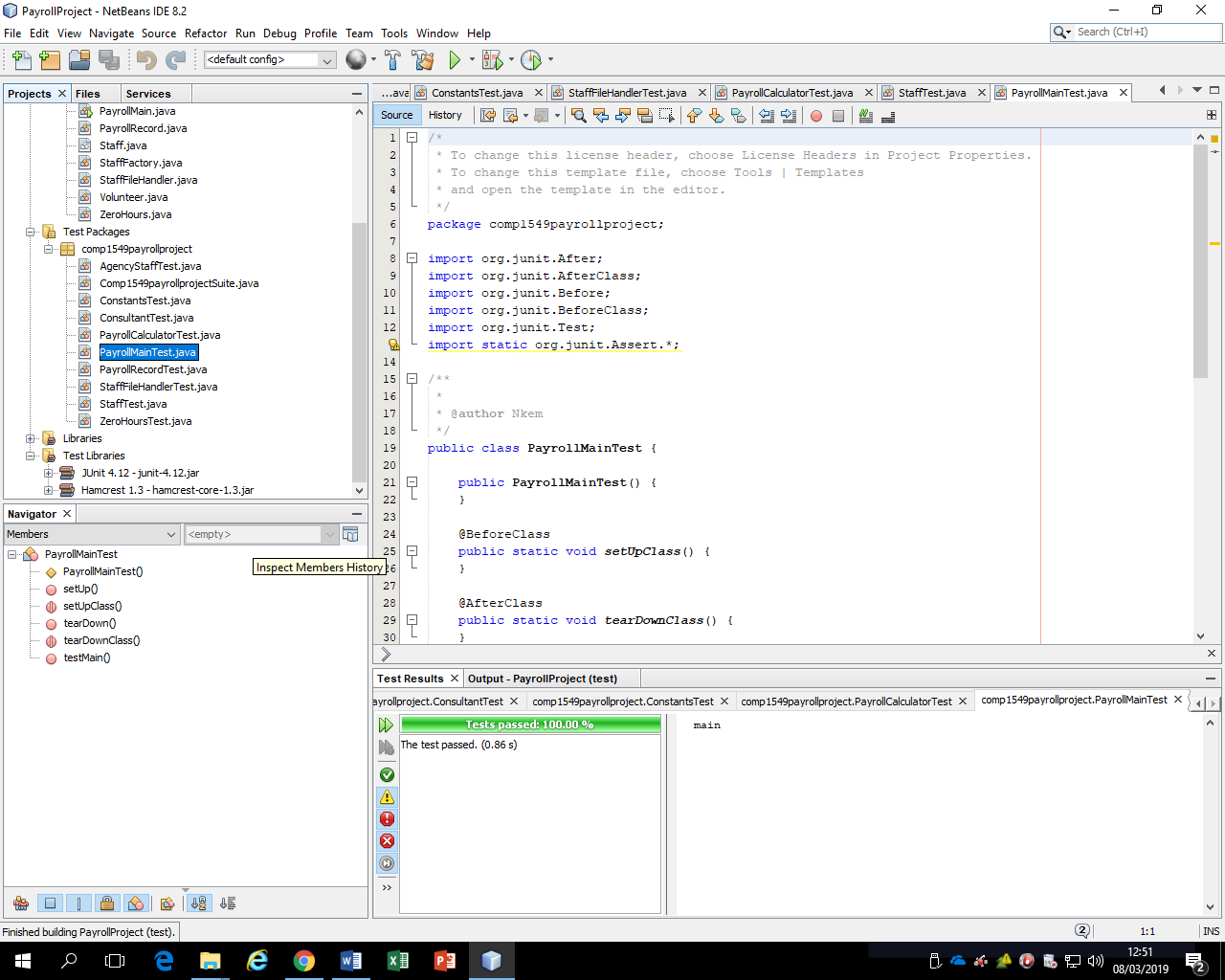
**Testing the files before adding the Volunteer and stafffactory classes- all passed**

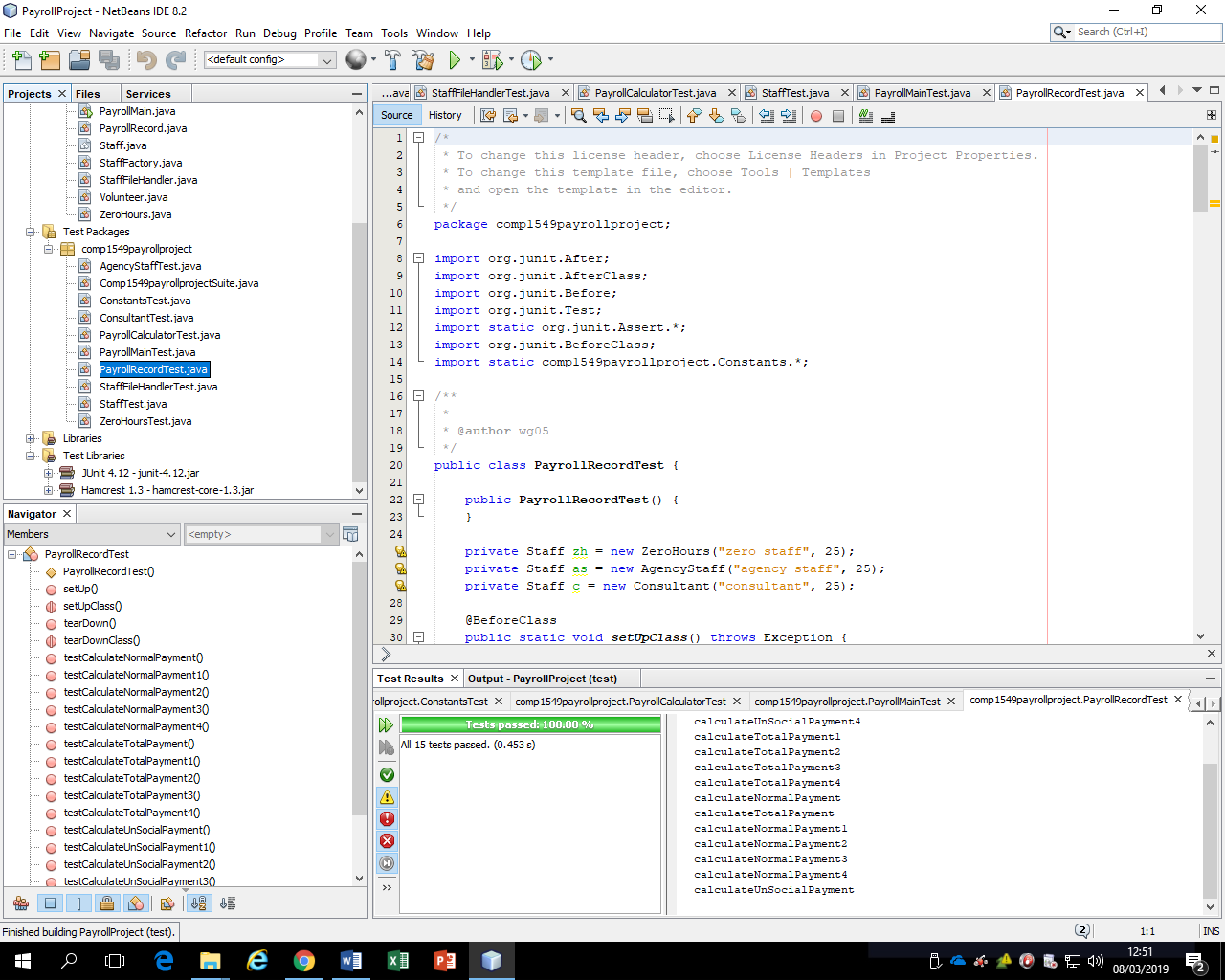


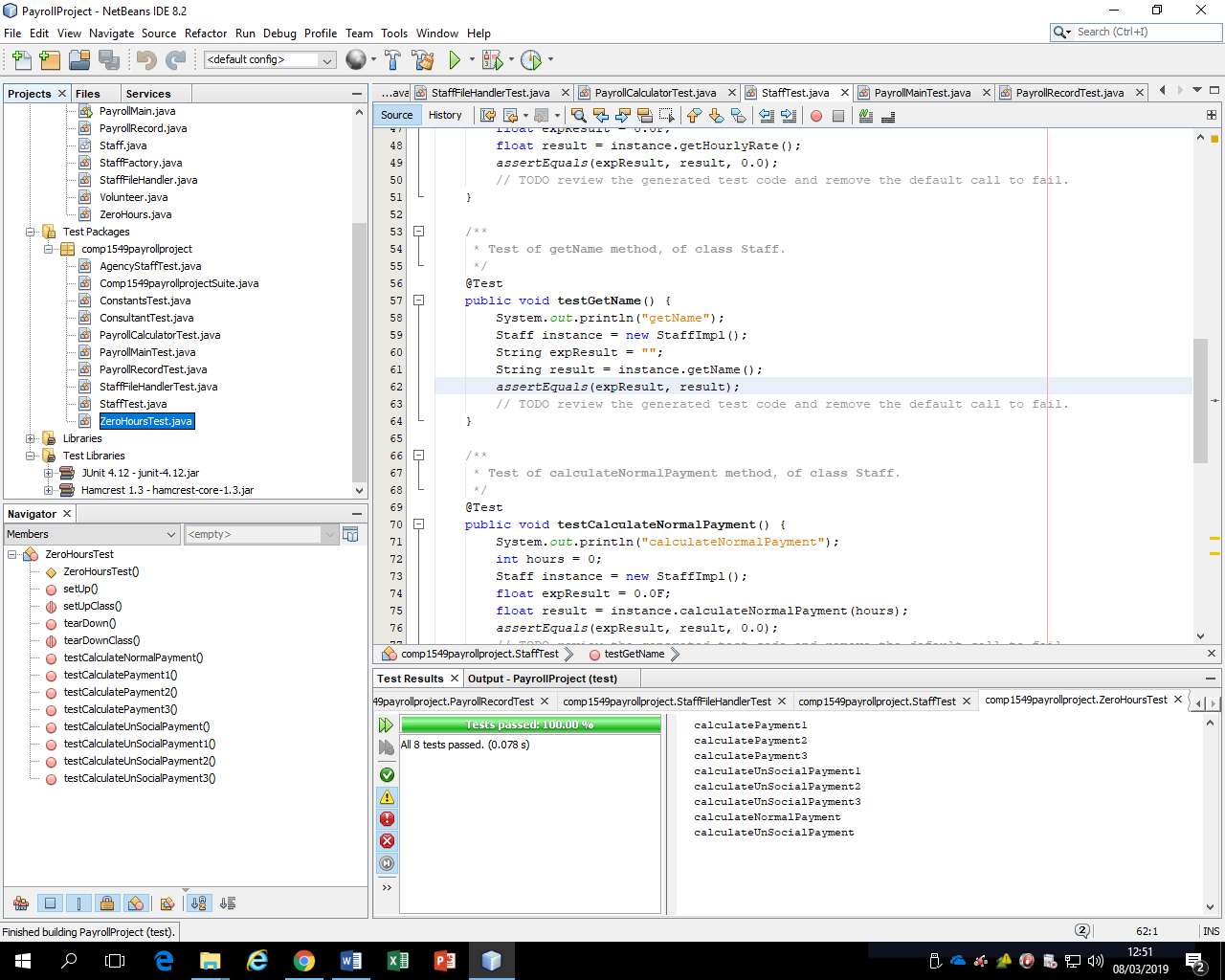


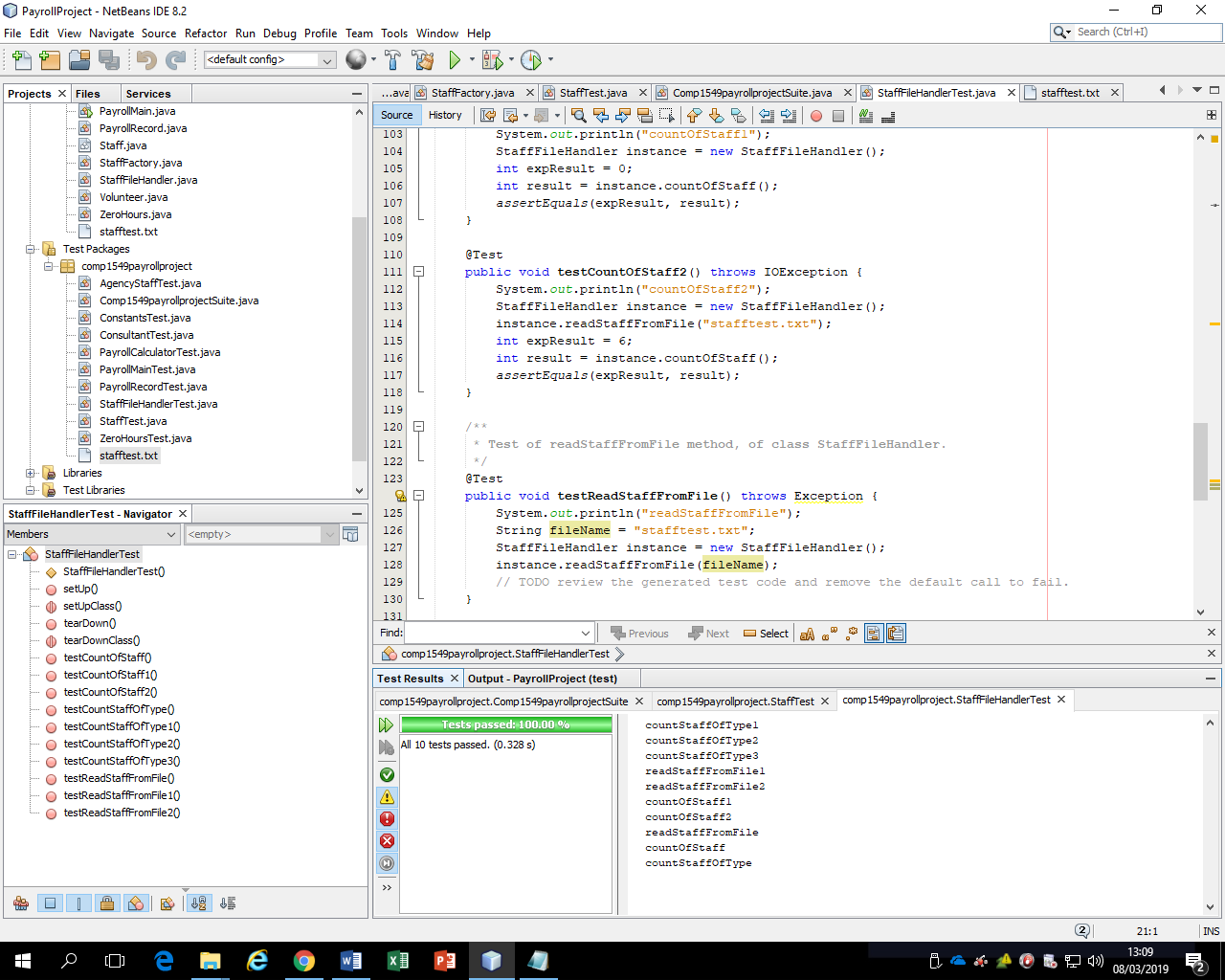


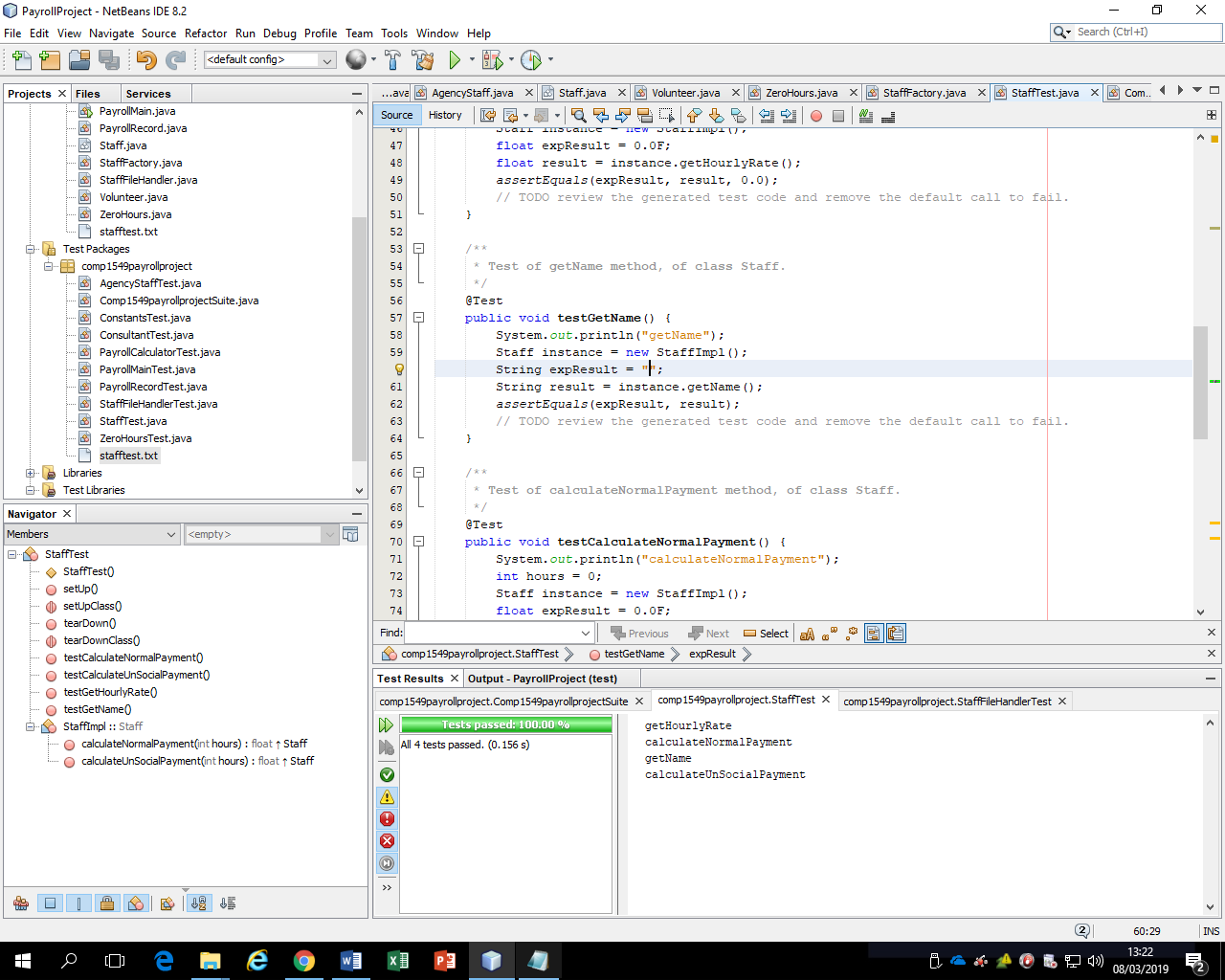


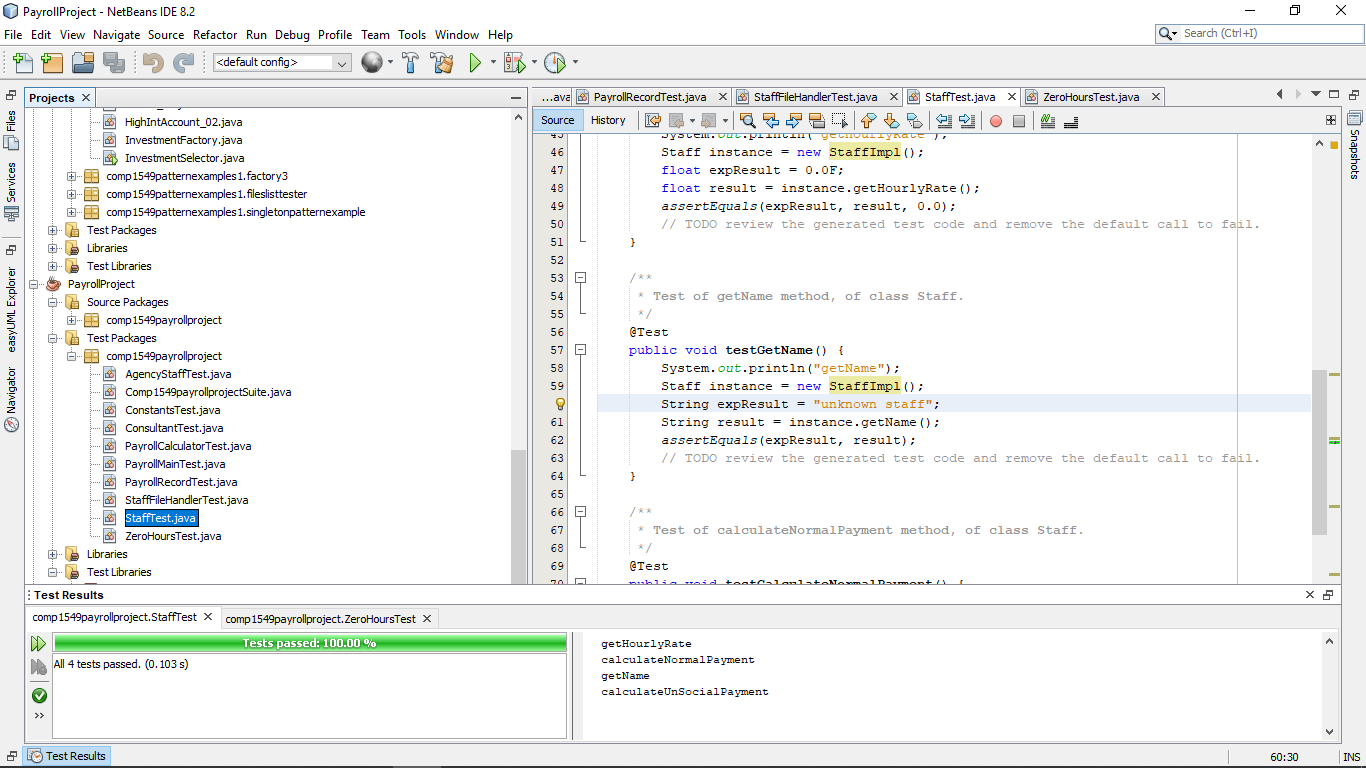


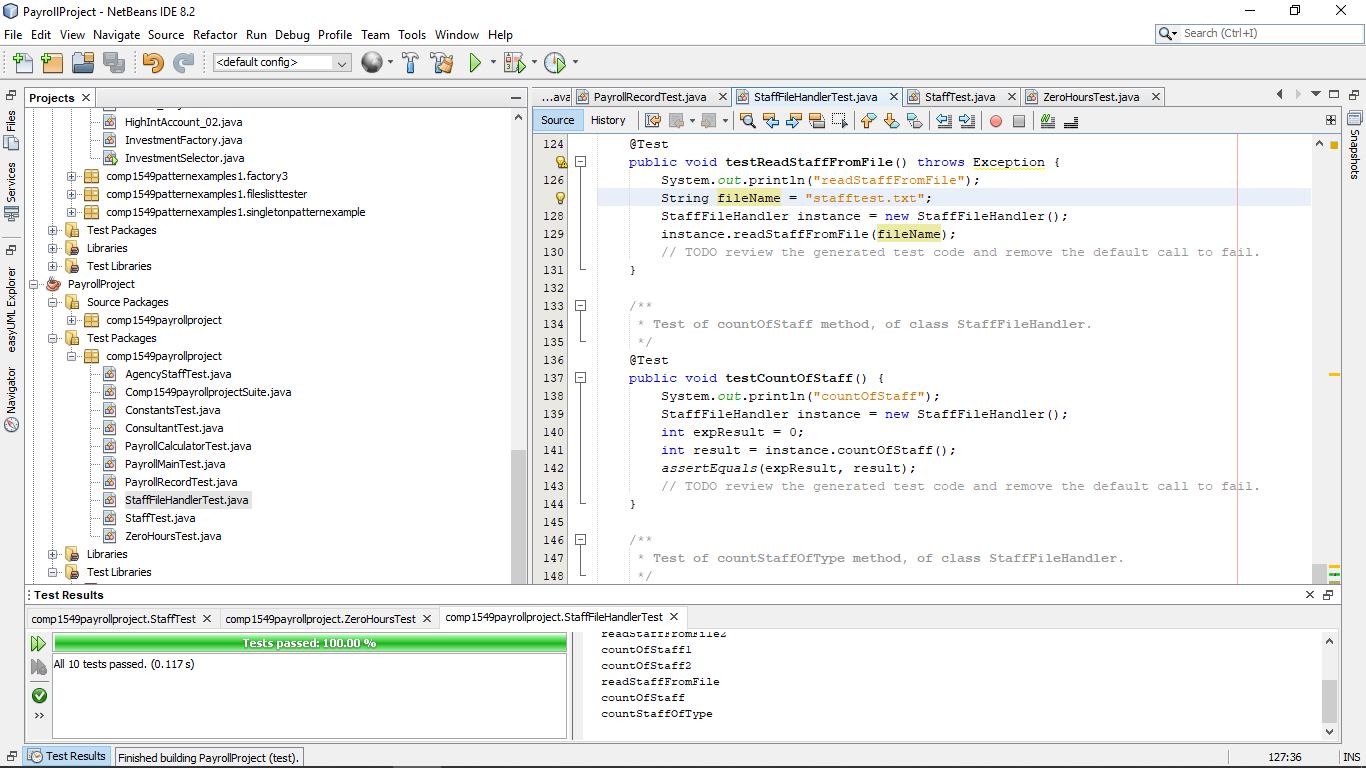


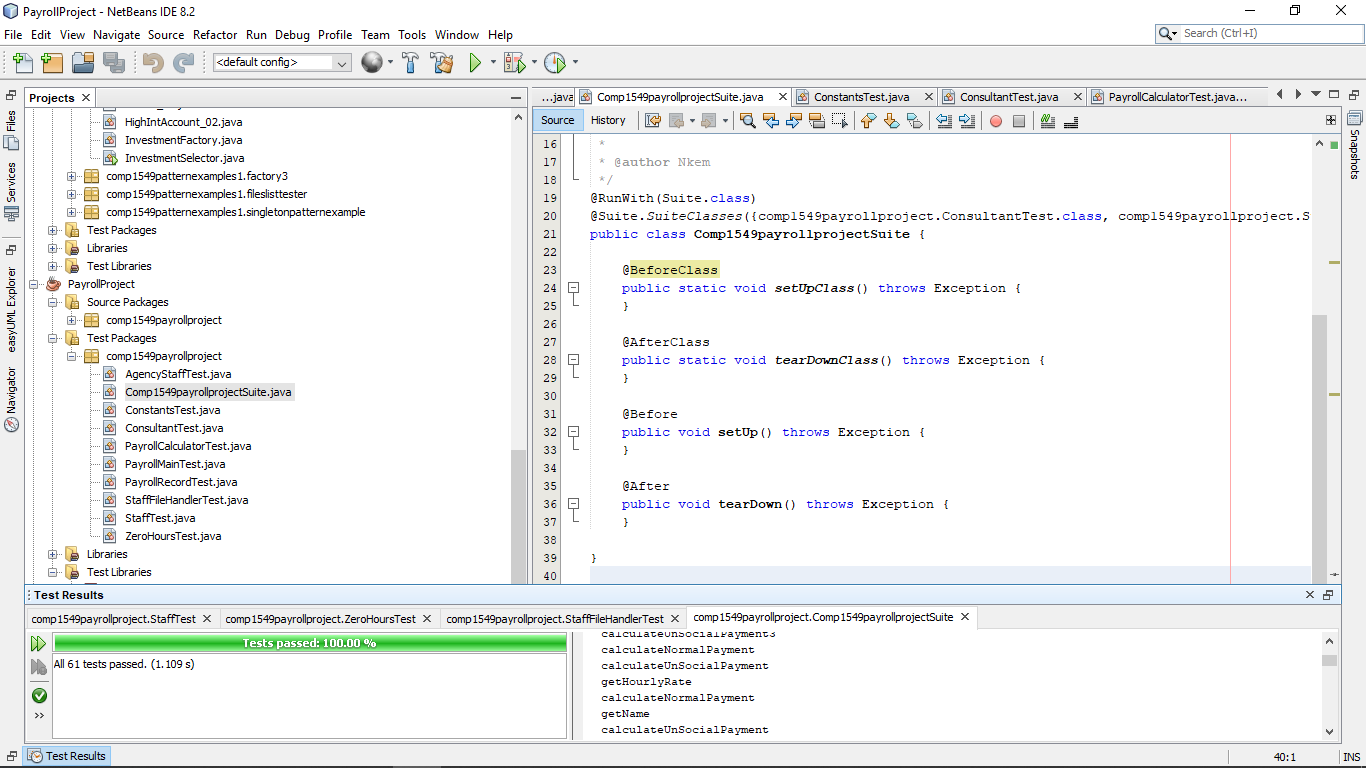












**Testing the files after adding the Volunteer and stafffactory classes- all passed**

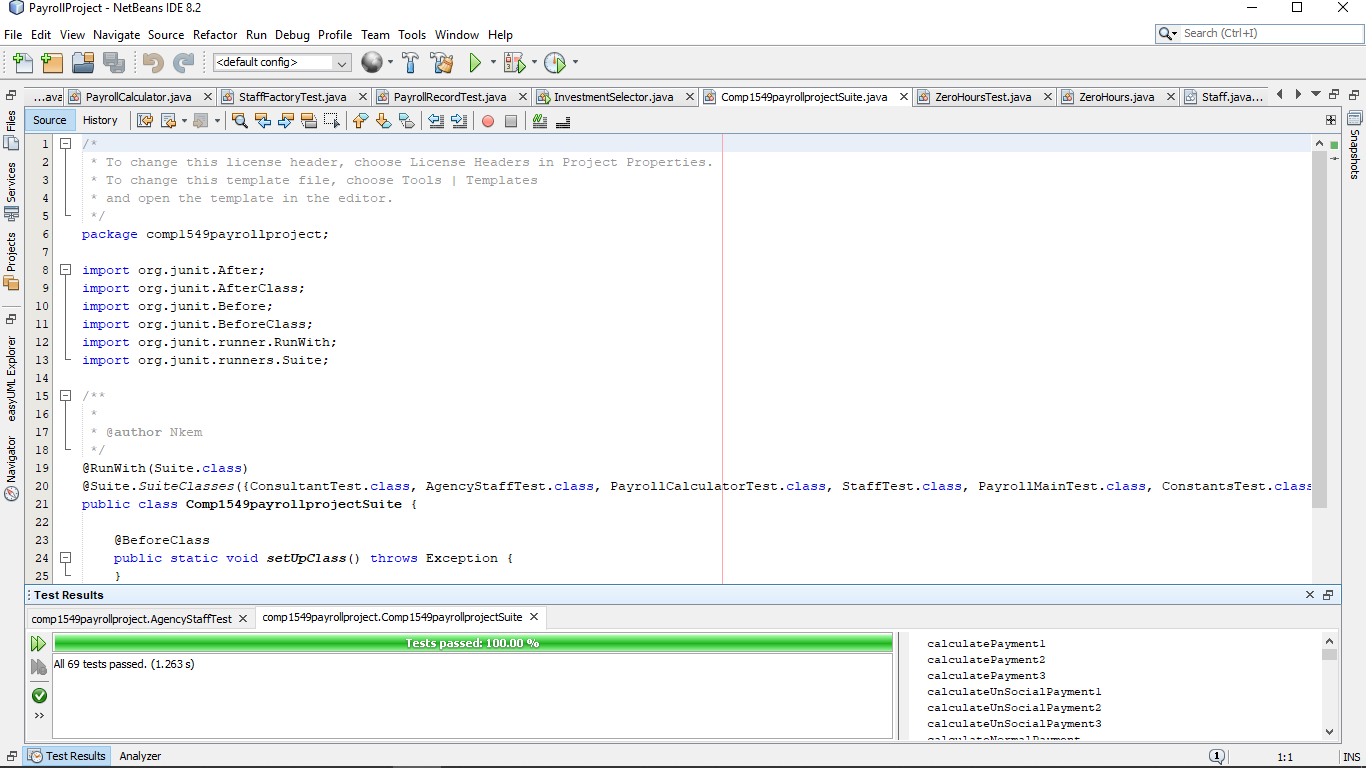


Figure 1Testing through the suite, now more tests have been added compared to before

2.2 Copy and paste **code that you wrote or amended**. Please **format** it nicely and **make it easy** for the tutor to see and read your code.

**From ZeroHoursTest**

@Test

public void testCalculatePayment4() {

System.out.println("calculatePayment3");

int hours = 5;

Staff instance = StaffFactory.createStaff("ZERO", "Rose", 12);

float expResult = 60.0F;

float result = instance.calculateNormalPayment(hours);

assertEquals(expResult, result, 0.0);

//test if factory still produces a correct result

}

**From StaffFactoryTest**

@Test

//test for null/blank constructor

public void testCreateStaff() {

System.out.println("createStaff");

String type = "";

String name = "";

float hourly\_rate = 0.0F;

Staff expResult = null;

Staff result = StaffFactory.createStaff(type, name, hourly\_rate);

assertEquals(expResult, result);

}

@Test

//testing if a method using a factory pattern created object works properly

public void testStaffMethod() {

System.out.println("createStaff");

String type = "ZERO";

String name = "Hal";

float hourly\_rate = 10.0F;

Staff exp = new ZeroHours("Hal", 10);

float expResult = exp.calculateNormalPayment(3);

Staff res = StaffFactory.createStaff(type, name, hourly\_rate);

float result = res.calculateNormalPayment(3);

assertEquals(expResult, result, 0.0);

}**From VolunteerTest**

/\*\*

\* Test of calculateNormalPayment method, of class Volunteer.

\*/

@Test

public void testCalculateNormalPayment() {

System.out.println("calculateNormalPayment");

int hours = 0;

Volunteer instance = new Volunteer();

float expResult = 0.0F;

float result = instance.calculateNormalPayment(hours);

assertEquals(expResult, result, 0.0);

// TODO review the generated test code and remove the default call to fail.

}

@Test

public void testCalculatePayment2() {

System.out.println("calculatePayment2");

int hours = 5;

Volunteer instance = new Volunteer("Johnny", 10);

float expResult = 0.0F;

float result = instance.calculateNormalPayment(hours);

assertEquals(expResult, result, 0.0);

}

/\*\*

\* Test of calculateUnSocialPayment method, of class Volunteer.

\*/

@Test

public void testCalculateUnSocialPayment() {

System.out.println("calculateUnSocialPayment");

int hours = 0;

Volunteer instance = new Volunteer();

float expResult = 0.0F;

float result = instance.calculateUnSocialPayment(hours);

assertEquals(expResult, result, 0.0);

}

@Test

public void testCalculateUnSocialPayment2() {

System.out.println("calculateUnSocialPayment");

int hours = 4;

Volunteer instance = new Volunteer("Jade", 8);

float expResult = 48.0F;

float result = instance.calculateUnSocialPayment(hours);

assertEquals(expResult, result, 0.0);

}

**Volunteer Class Code**

//creation of volunteer class that is a subclass of staff

public class Volunteer extends Staff {

//default constructor for volunteer

public Volunteer() {

this("unknown volunteers", 0);

}

/\*Basic constructor using a name parameter for the volunteer, this variable is

from staff, and a float hourlyrate for the hourlyrate for volunteer

\*/

public Volunteer(String name, float hourlyRate) {

super(name, hourlyRate);

}

//volunteers do not have normal payment, so this should always return zero

@Override

public float calculateNormalPayment(int hours) {

return 0.0F;

}

/\*this multiplies the hourly rate with the number of hours(this parameter)

and the unsocial\_rate\_multiplier inherited from Constants

which is then returned

\*/

@Override

public float calculateUnSocialPayment(int hours) {

return getHourlyRate() \* hours \* UNSOCIAL\_RATE\_MULTIPLIER;

}

}

**Failed PayrollCalculator class’ add payroll method**

public void addPayrollRecord(String name, float hourlyRate, STAFF\_TYPE type, int normalHours, int unsocialHours) {

//modified the code to instead call the factory class in order to create

//a new instance of a staff subclass, which is also added to the payroll record,

Staff newStaff = StaffFactory.createStaff(type.toString() , name, hourlyRate);

PayrollRecord newPayrollRecord;

newPayrollRecord = new PayrollRecord(newStaff, normalHours, unsocialHours);

payrollRecords.add(newPayrollRecord);

}

Due to some errors with testing blank strings with the PayrollCalculator, I had to add a nullpointerargumentexception via a try-catch in order to allow for null values to be recognised and not return an error with the test

**StaffFactory-original class**

public class PayrollCalculator {

private List<PayrollRecord> payrollRecords;

/\*\*

\* Default constructor creates and empty list of payroll records

\*/

public PayrollCalculator() {

this.payrollRecords = new ArrayList<>();

}

/\*\*

\* Creates a payroll record and adds it to the list of payroll records

\*

\* @param name full name of the member of staff

\* @param hourlyRate default hourly payment rate for the member of staff

\* @param type the type of staff member

\* @param normalHours number of hours worked by the staff to be paid at

\* their normal hourly rate

\* @param unsocialHours number of hours worked by the member of staff at

\* their unsocial hourly rate

\*/

public void addPayrollRecord(String name, float hourlyRate, STAFF\_TYPE type, int normalHours, int unsocialHours) {

//modified the code to instead call the factory class in order to create

//a new instance of a staff subclass, which is also added to the payroll record,

Staff newStaff;

PayrollRecord newPayrollRecord;

//try catch designed to handle if the type entered is null

try

{

newStaff = StaffFactory.createStaff(type.toString() , name, hourlyRate);

newPayrollRecord = new PayrollRecord(newStaff, normalHours, unsocialHours);

payrollRecords.add(newPayrollRecord);

}

//notes a nullpointerexception if there is a blank or null parameter in

catch(NullPointerException ex)

{

System.out.println("NullPointerexception has occured");

}

}

/\*\*

\* Calculate and return the total current payroll bill

\*

\* @return the current total payroll bill including both normal and unsocial

\* hours worked based on all the records in the payroll records list.

\*/

public float getPayrollTotal() {

float total = 0.0F;

for (PayrollRecord pr : payrollRecords) { // loop through all the records in the list

total += pr.calculateTotalPayment();

}

return total;

}

Similar to the Payroll calculator, an IllegalArgumentException happened with a null “type” string for the constructor for the Factory class, leading to issues with testing the factory class. To rectify this, we added a try-catch for the creation of the staff subclasses via factory pattern, as well as refactoring the structure of the class to support this change

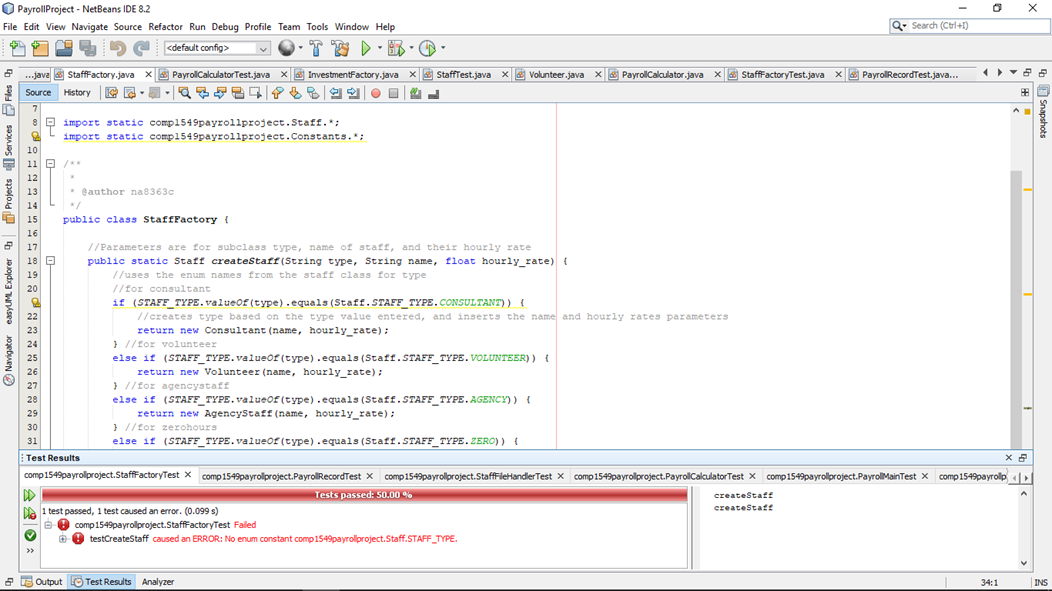


Figure 2Example of the general error I had with this class when testing

**StaffFactory class- refactored**

public class StaffFactory {

//Parameters are for subclass type, name of staff, and their hourly rate

public static Staff createStaff(String type, String name, float hourly\_rate) {

//declares null abstract object that will use polymorphism alongside the factory pattern

Staff newStaff = null;

//uses the enum names from the staff class for type

//for consultant

//try-catch set up

try{

if (STAFF\_TYPE.valueOf(type).equals(Staff.STAFF\_TYPE.CONSULTANT)) {

//initialises the newStaff class as one the enum values entered through the “type” string parameter, and inserts the name and hourly rates parameters

newStaff= new Consultant(name, hourly\_rate);

} //for volunteer

else if (STAFF\_TYPE.valueOf(type).equals(Staff.STAFF\_TYPE.VOLUNTEER)) {

newStaff= new Volunteer(name, hourly\_rate);

} //for agencystaff

else if (STAFF\_TYPE.valueOf(type).equals(Staff.STAFF\_TYPE.AGENCY)) {

newStaff= new AgencyStaff(name, hourly\_rate);

} //for zerohours

else if (STAFF\_TYPE.valueOf(type).equals(Staff.STAFF\_TYPE.ZERO)) {

newStaff= new ZeroHours(name, hourly\_rate);

} /\*if nothing valid is inserted as a parameter, then it returns null through the new catch created to note if the illegalargument exception happens\*/

}

catch(IllegalArgumentException e){

}

//return newStaff

return newStaff;

}

**Logbook 4 – "Creating Software Components"**

**Part B – Software Development:**

**Section 1 – Brief statement of levels you have completed (same for both partners)**

|  |  |
| --- | --- |
| Which type of dashboard application did you implement? | Train small air craft |
| 1.1 Circle the parts of the coursework you have fully completed and are fully working | 1a 1b 2a 2bi 2bii 3  4a 4bi 4bii 5a 5b 6a 6b |
| 1.2 Circle the parts of the coursework you have partly completed or are partly working | 1a 1b 2a 2bi 2bii 3  4a 4bi 4bii 5a 5b 6a 6b |
| Briefly explain your answer if you circled any parts in 1.2 | |

**Section 2 – UML class diagram of your application (same for both team members)**

Include a UML class diagram of your application. Try to use correct notation and make it an accurate representation of what you have implemented. Include enough detail to show the important aspects of the design but not so detailed that the important information is lost in a mass of detail.

**Section 3 – Concise list of bugs and weaknesses (same for both team members)**

**3.1 Bugs - List plus brief description**

**3.2 Weaknesses - List plus brief description**

**Section 4 - Documentation of each of the levels you completed (same for both team members)**

**4.1 Level 1 -** Brief description (up to 200 words) of the ways in which your program is more reliable, flexible and easy to use than the example code you were supplied. Please include word count.

**4.2 Level 2**

4.2.1 Give a UML class diagram for the inheritance hierarchy(s) you implemented. This may be an extract of your earlier UML diagram. The only details needed are the classes and the relationships between them.

4.2.2 Give a UML class diagram for the interfaces. This may be an extract of your earlier UML diagram. The only details needed are the interfaces and the classes that implement them.

4.2.3 Brief reflection (up to 300 words) on how the object-oriented features you implemented improve the design of the system in order to make it easier to maintain and enhance in the future. Please include word count.

**4.3 Level 3**

List of classes you tested using JUnit and the number of tests for each class

**4.4.1 Pattern 1**

Name: *Singleton*

UML class diagram showing the classes and interfaces involved in your implementation of the pattern and the relationships between them.

Brief reflection (up to 150 words) about how this design pattern has improved the design of the system in order to make it easier to maintain and enhance in the future. Please include word count.

**4.4.2 Pattern 2**

Name:

UML class diagram showing the classes and interfaces involved in your implementation of the pattern and the relationships between them.

Brief reflection (up to 150 words) about how this design pattern has improved the design of the system in order to make it easier to maintain and enhance in the future. Please include word count.

**4.4.3 Pattern 3**

Name:

UML class diagram showing the classes and interfaces involved in your implementation of the pattern and the relationships between them.

Brief reflection (up to 150 words) about how this design pattern has improved the design of the system in order to make it easier to maintain and enhance in the future. Please include word count.

**4.5 Level 5**

**4.5.1 Component 1**

Classname:

A brief (up to 50 words) description of its purpose and role.

Number of properties you have coded for the component

Does the BeanInfo file cause the icon to be displayed?

**4.5.2 Component 2**

Classname:

A brief (up to 50 words) description of its purpose and role.

Number of properties you have coded for the component

Does the BeanInfo file cause the icon to be displayed?

**4.6 Level 6**

4.6.1 Stretch Feature a)

Brief description of the feature (up to 200 words)

List of strengths and weaknesses of your implementation of the feature

4.6.2 Stretch Feature b)

Brief description of the feature (up to 200 words)

List of strengths and weaknesses of your implementation of the feature

**Section 5 – Annotated screen shots demonstrating each of the stages that you have completed (same for both team members)**

**Section 6 - 350-500 words reflecting on pair programming (write this section individually)**

Carefully read the coursework specification which tells you what is required for this section.