CS4001NI-Programming

30% Individual Coursework

2020-21 Autumn

Student name: Ishan Gurung

London Met ID: 19031315

College ID: NP01NT4A190139

Assignment Due date:5th june,2020.

Assignment submission Date:5th june,2020.

Contents

Introduction

In week 20 our coursework was released and the coursework is all about creating the (GUI) graphic user interface for hiring staff about we learned in this semester. So, to create the GUI form given textfields which is given in the specific name in the coursework and to do this assignment we should have the basic knowledge about Jframe, Jtextfiel, Jbuttom and how they work when the button is pressed. And also, must know about the try-catch block catching and throwing exceptions. So, in this coursework we made (GUI) graphic user interface by linking the previous coursework in new INGNepal class. The main motive of INGNepal class is to initialize functionality to various component of the GUI. The INGNepal main class links to two sub class they are fulltime staffHire and parttime staffhire. Similarly by using array list it also links with the reference datatype as created in INGNepal class. It runs to appoints the staff of the full time and part time staff. It also terminate the staff when required. It display all information about the appointed staff and clear all the value entered from the fields.

For this program we were suggested to use bluej for the code and for the report writing we were suggested to do in ms-word by our module leader. Bluej is the application that helps to develop programs quickly and easily.

Class diagram

Class Diagram is a tabular description of all the methods and variables used to write codes for a unique class. It contains three extraordinary parts, the second instance variables in the title of the first category and eventually in the process of the series. We have to use them to manipulate modifiers that are used with methods and variables of opportunity. There is sign convention for class diagram like “+” for private variables and “– “for access modifier. Although, there are more sign conventions too.

Class diagram

INGNepal
- JFrame frm;
- JLabel title;
- JTextField;
- JComboBox;
- JButton;
+ FullTimeStaffHire
+ fullTimeStaffHire
+ PartTimeStaffHire
+ FullTimeStaffHire
+ PartTimeStaffHire
+ FullTimeStaffHire
+ PartTimeStaffHire
+ void main
+ void actionPerformed(ActionEvent e)
+ void addforfulltimestaffhire()
+ void appointforfulltimestaffhire()
+ void addforParttimestaffhire()
+ void appointforparttimestaffhire()
+ void Terminate()
+ void Display()
+ void Clear()
+ static void main(String[] args)

Pseudocode

START

DECLARE CLASS VARIABLE

```
  JFrame frm;

        JLabel lblVacancyNumber, JLabel lblDesignation,JLabellblJobType, JLabel lblSalary,
        JLabel lblWorkingHour, JLabellblVacancyNumberApnt,  JLabellblStaffName, JLabel
        lblJoinDate, JLabellblQualification, JLabel  lblApointedBy;


  JTextField txtVacancyNumber, JTextField  txtDesignation, JTextField txtJobType, JTextField
txtSalary,    JTextField txtWorkingHour, JTextField txtVacancyNumberApnt,       JTextField
txtStaffName, JTextField txtJoinDate, JTextField txtQualification, JTextField  txtApointedBy;


JcomboBox cmbJobType , cmbJobType;

Jbutton btnAddFullTimeVacancy, btnAddPartTimeVacancy, btnAppointFT,
btnAppointFT1,btnDisplay,btnClear,btnTerminate;    String VacancyNumber;
```

DO

```
        BUILD A METHOD StaffHireForm
        INITIALIZE The JFrame  StaffHire
        GIVE SIZE OF THE FRAME (800,600)
        GIVE LAYOUT OF THE FRAME
        GIVE VISIBLITY OF THE FRAME

        Jlabel fulltime StaffHire as title
        Set the setbounds
        Add the form


        CREATE JLabel : labelVacancyNumber
       SETTING Bounds of the JLabel
        ADDING frame to the JLabel
```

```
CREATE JTextField : txtVacancyNumber
SETTING Bounds of the JTextField
ADDING frame to the JTextField

CREATE JLabel : labelDesignation
SETTING Bounds of the JLabel
ADDING frame to the JLabel
CREATE JTextField : txtDesignation
SETTING Bounds of the JTextField
ADDING frame to the JTextField

CREATE JLabel : labelJobType
SETTING Bounds of the JLabel
ADDING frame to the JLabel
CREATE String JobType
CREATE ComboBox :cmbJobType
SETTING Bounds of JComboBox
ADDING frame to the JComboBoox

CREATE JLabel : labelSalary
SETTING Bounds of the JLabel
ADDING frame to the JLabel
CREATE JTextField : txtSalary
SETTING Bounds of the JTextField
ADDING frame to the JTextField


CREATE JLabel : labelWorkingHour
SETTING Bounds of the JLabel
ADDING frame to the JLabel
```

```
CREATE JTextField : txtWorkingHour
SETTING Bounds of the JTextField
ADDING frame to the JTextField



CREATE JBUTTON : btnAddFullTimeVacancy
SETTING Bounds of the JTextField
ADDING ActionListener: btnAddFullTimeVacancy
ADDING frame to the JTextField

CREATE JLabel : labelQualification
SETTING Bounds of the JLabel
ADDING frame to the JLabel
CREATE JTextField : txtQualification
SETTING Bounds of the JTextField
ADDING frame to the JTextField

CREATE JLabel : labelStaffName
SETTING Bounds of the JLabel
ADDING frame to the JLabel
CREATE JTextField : txtStaffName
SETTING Bounds of the JTextField
ADDING frame to the JTextField

CREATE JLabel : labelJoinDate
SETTING Bounds of the JLabel
ADDING frame to the JLabel
CREATE JTextField : txtJoinDate
SETTING Bounds of the JTextField
ADDING frame to the JTextField
```

```
CREATE JLabel : labelWorkingShifts
SETTING Bounds of the JLabel
ADDING frame to the JLabel
CREATE JTextField : txtWorkingshifts
SETTING Bounds of the JTextField
ADDING frame to the JTextField

CREATE JLabel : labelWagesPerHour
SETTING Bounds of the JLabel
ADDING frame to the JLabel
CREATE JTextField : txtWagesPerHour
SETTING Bounds of the JTextField
ADDING frame to the JTextField

CREATE JButton : btnAppointFT
SETTING Bounds of the JTextField
ADDING ActionListener : btnAppointFT
ADDING frame to the JTextField


Jlabel parttime StaffHire as title
Seting  the setbound of the tittle
Add the form


CREATE JLabel : labelVacancyNumber
SETTING Bounds of the JLabel
ADDING frame to the JLabel
CREATE JTextField : txtVacancyNumber
```

```
SETTING Bounds of the JTextField
ADDING frame to the JTextField

CREATE JLabel : labelDesignation
SETTING Bounds of the JLabel
ADDING frame to the JLabel

CREATE JTextField : txtDesignation
SETTING Bounds of the JTextField
ADDING frame to the JTextField

CREATE JLabel : labelJobType
SETTING Bounds of the JLabel
ADDING frame to the JLabel

CREATE String JobType
CREATE ComboBox :cmbJobType
SETTING Bounds of JComboBox
ADDING frame to the JComboBoox

CREATE JLabel : labelSalary
SETTING Bounds of the JLabel
ADDING frame to the JLabel

CREATE JTextField : txtSalary
SETTING Bounds of the JTextField
ADDING frame to the JTextField


CREATE JLabel : labelWorkingHour
```

```
SETTING Bounds of the JLabel
ADDING frame to the JLabel
CREATE JTextField : txtWorkingHour
SETTING Bounds of the JTextField
ADDING frame to the JTextField


CREATE JBUTTON: btnAddpartTimeVacancy
SETTING Bounds of the JTextField
ADDING ActionListener: btnAddpartTimeVacancy
ADDING frame to the JTextField

CREATE JLabel : labelQualification
SETTING Bounds of the JLabel
ADDING frame to the JLabel
CREATE JTextField : txtQualification
SETTING Bounds of the JTextField
ADDING frame to the JTextField

CREATE JLabel : labelStaffName
SETTING Bounds of the JLabel
ADDING frame to the JLabel

CREATE JTextField : txtStaffName
SETTING Bounds of the JTextField
ADDING frame to the JTextField

CREATE JLabel:j labelJoinDate
SETTING Bounds of the JLabel
ADDING frame to the JLabel
```

```
CREATE JTextField : txtJoinDate
SETTING Bounds of the JTextField
ADDING frame to the JTextField

CREATE JLabel : labelWorkingShifts
SETTING Bounds of the JLabel
ADDING frame to the JLabel

CREATE JTextField : txtWorkingshifts
SETTING Bounds of the JTextField
ADDING frame to the JTextField

CREATE JLabel : labelWagesPerHour
SETTING Bounds of the JLabel
ADDING frame to the JLabel

CREATE JTextField : txtWagesPerHour
SETTING Bounds of the JTextField
ADDING frame to the JTextField

CREATE JButton : btnAppointFT
SETTING Bounds of the JTextField
ADDING ActionListener: btnAppointFT
ADDING frame to the JTextField

Create Jbutton: btnDisplay
Setting set bound of Jtextfield
Add frame to Jtextfield
```

```
    Create Jbutton: btnTeminate
    Setting set bound of Jtextfield
    Add frame to Jtextfield


    Create Jbutton: btnclear
    Setting set bound of Jtextfield
    Add frame to Jtextfield


CREATE method actionPerformed (ActionEvent e)
    if (e.getSource()==btnAddFullTimeVacancy)
    addFullTimeVacancy

    if (e.getSource()==btnAddPartTimeVacancy)
       addPartTimeVacancy

    else if(e.getSource()==btnAppointFT)
    appointFullTime

    else if(e.getSource()==btnDisplay)
    Display

    else if(e.getSource()==btnTerminate)
    Terminate
    else if(e.getSource()==btnClear)
    Clear

FUNCTION: btnaddForFullTimeStaffHire()
    TRY
```

```
            int vno=Integer.parseInt(txtVacancyNumber.getText());
            String designation=txtDesignation.getText();
            int salary=Integer.parseInt(txtsalary.getText());
            int workingHour=Integer.parseInt(txtworkingHour.getText());
            String JobType=(cmbJobType.getSelectedItem()).toString();


            boolean isDuplicateVacancyno=false;
      FOR
            IF (var.getVacancynumber()==Vno)
                  isDuplicateVacancyno=false;
      END
            IF (isDuplicateVacancyno==false)
                  FullTimeStaffHire obj=new FullTimeStaffHire(vno,
                  desgination,jobType,salary,workingHour);
                   list.add(obj);
                  JOptionPane.showMessageDialog(frm,"vacancy added "");
            ELSE
                  JOptionPane.showMessageDialog(frm,"Input Vacancy no is already in the
                  list. "+list.size())
      END
            CATCH (Exception exp)
                  JOptionPane.showMessageDialog(frm,"Input is invalid. "");
      END CATCH


FUNCTION: btnAppointFT()
      int vno = Integer.parseInt(txtVacancyNumberApnt.getText());
     String staffName=txtStaffName.getText();
      String joinDate=txtJoinDate.getText();
      String qualification=txtQualification.getText();
      String appointedBy=txtApointedBy.getText();
     boolean vnoFound=false;
```

```
FOR
    IF (obj.getVacancynumber()==Vno)
        VnoFound=true;
    END
        IF (obj instanceof FullTimeStaffHire)
        FullTimeStaffHire h = (FullTimeStaffHire)obj;
            IF (h1.getappointedBy()==true)
                JOptionPane.showMessageDialog(frm,"Staff already Hired");
                break;
            ELSE

        h1.hireFullTimeStaff(StaffName,JoiningDate,Qualification,AppointedBy);
                JOptionPane.showMessageDialog(frm,"Staff has already hired");


                 break;
            ENDIF
            ELSE
                JOptionPane.showMessageDialog(frm,"it is not for part time");
                break;
    FUNCTION: Display()
    FOR (obj:list)
        IF (obj instanceof FullTimeStaffHire)
            obj=(FullTimeStaffHire)obj;
              obj.display();
           else if(obj instanceof PartTimeStaffHire)
        PartTimeStaffHire obj11 =(PartTimeStaffHire)obj;
        obj11.display();
        END


FUNCTION: Terminate()
```

```
int Vno=Integer.parseInt(txt1partterminated.getText());

    IF (obj.getVacancynumber()==Vno)

        IF (obj instanceof PartTimeStaffHire)
            PartTimeStaffHire hre=(PartTimeStaffHire)obj;

            IF (hre getterminated()==true)
            JOptionPane.showMessageDialog(frm,"Now, Part Time Staff is
            successfully terminated");
            break;
            ELSEIF
                hre.getterminated()==true;
                JOptionPane.showMessageDialog("frm,"Sorry!Staff has
                already been terminated");
                 break;
            else

        hre.setterminated();
        JOptionPane.showMessageDialog(frm,"Part time terminated");
        break;
      else
        JOptionPane.showMessageDialog(frm,"Not for part time");
        break;
```

```
FUNCTION: btn1Clear()
            txtVacancyNumber.setText(" ");
    txtDesignation.setText(" ");
    cmbJobType.setSelectedIndex(0);
    txtSalary.setText(" ");
    txtWorkingHour.setText(" ");


    txtVacancyNumberApnt.setText(" ");
    txtStaffName.setText(" ");
    txtJoinDate.setText(" ");
    txtQualification.setText(" ");
    txtApointedBy.setText(" ");


    txtVacancyNumber1.setText(" ");
    txtDesignation1.setText(" ");
    cmbJobType1.setSelectedIndex(0);
    txtshifts.setText(" ");
    txtSalary1.setText(" ");
    txtWorkingHour1.setText(" ");


    txtVacancyNumberApnt1.setText(" ");
    txtStaffName1.setText(" ");
    txtJoinDate1.setText(" ");
    txtQualification1.setText(" ");
    txtApointedBy1.setText(" ");
    Display JOptionPane.showMessageDialog(frm,"Text fields cleared");
    END

FUNCTION: addForPartTimeStaffHire()
    TRY
    int vacancynumber=Integer.parseInt(txtVacancyNumber1.getText());
    String designation=txtDesignation1.getText();
    int wagesperhour=Integer.parseInt(txtSalary1.getText());
    int workingHour=Integer.parseInt(txtWorkingHour1.getText());
    String jobType=(cmbJobType1.getSelectedItem()).toString();
    String shifts=txtshifts.getText();
    boolean isDuplicateVno=false;
  FOR
        IF (var.getVacancynumber()==vacancy number)
              isDuplicateVno=false;
              break;
  END
        IF (isDuplicateVno==false)
              PartTimeStaffHire obj=new PartTimeStaffHire
              (vacancynumber,designation,jobType,wagesperhour,workingHour,shifts);
                list.add(obj);
               JOptionPane.showMessageDialog(frm,"vacancy added");
        ELSE
              JOptionPane.showMessageDialog(panel1,"Input Vacancy no is already in
              the list. "+list.size()));
        END
              CATCH (Exception exp)
                    JOptionPane.showMessageDialog(panel1,"Input is invalid. ");
        END CATCH
```

FUNCTION: btn1AppointPT()

```
 int vno=Integer.parseInt(txtVacancyNumberApnt.getText());
       String staffName=txtStaffName.getText();
       String joinDate=txtJoinDate.getText();
       String qualification=txtQualification.getText();
       String appointedBy=txtApointedBy.getText();
```

```
    boolean vnoFounds=false;
FOR
    IF (obj.getVacancynumber()==vno)
        VnoFound=true;
    END
        IF (obj instanceof PartTimeStaffHire)
        PartTimeStaffHire h=(PartTimeStaffHire)obj;
         if(h.getJoined()==true)
        JOptionPane.showMessageDialog(frame,"Staff Has Hired");
         break;
            ELSE
                PartTimeStaffHire h=(PartTimeStaffHire)obj;
                (StaffName,JoiningDate,Qualification,AppointedBy);
                 JOptionPane.showMessageDialog(panel,"Staff has been hired");
                break;
    ENDIF
            ELSE
                JOptionPane.showMessageDialog(frame," not for fulltime staff
                Hire ");
                 break;
            ENDIF
        ELSE
            JOptionPane.showMessageDialog(frame,"Invalid Vacancy");
            break;
```

FUNCTION: Main

```
    INGNepal A=new INGNepal();
    A.StaffHireForm();
```

Method description

a. Method name: void staffHireForm()

this method is a mutator method. It has void return type. This is the function to create the graphical user interface (GUI) by containing every variable in a container. It is also called as the container which contains JLabel, JTextField, Jbutton, JcomboBox,etc

b. Method name: void actionPerformed(ActionEvent e)

It is also mutator method which has void return type. This method is used to for working in the button or clarify, when the button is clicked what should it do.

c. Method name: void addforFullTimeStaffHire()

This method is also a mutator method. It has void return type. The method addforFullTimeStaffHire() is made to created to entered the values in the textfied in the fulltime. When the users input the data in the textfied of fulltimestaffhire and stored the data of given textfield when it is clicked in addforFullTimeStaffHire().

d. Method name: void appointForFullTimeStaffHire()

This method is also a mutator method. It has void return type. The method appointForFullTimeStaffHire() is made to appointe the data that entered the values in the textfied in the fulltime. When the users input the data in the textfied of fulltimestaffhire and stored the data of given textfield when it is clicked in appointForFullTimeStaffHire() and appointed the full time staff.

e. Method name: void addforparttimeStaffHire()

it is also same as addforFullTimeStaffHire() which is mutator method which has void return type. So in the addforparttimeStaffHire() is made to create to entered the values in the textfied in the parttime. When the users input the data in the textfied of parttimestaffhire and stored the data of given textfield when it is clicked in addforpartTimeStaffHire().

f.**Method Name:** void appointForPartTimeStaffHire()

It is also same as addforFullTimeStaffHire() which is mutator method which has void return type.so in the appointForPartTimeStaffHire() is made to appointed the data that entered the values in the textfied in the parttime. When the users input the data in the textfied of parttimestaffhire and stored the data of given textfield when it is clicked in appointForpartTimeStaffHire() and appointed the full time staff.

g. Method Name: void terminated()

This method also has “set” keyword which represents so called mutator method.it has void types of return types. The reason for creating this method to access ever object and gives output as per condition and termination.

h. Method Name: void display ()

This method is also a mutator method. It has void return type. This method is created in order to print out the information of the objects of this class. This method displays all the textfield data which are stored in the INGNepal class when the display is clicked.

I. **Method Name:** void Clear()

This method is also a mutator method. It has void return type. The method is made to clear all the input which is fillend in JText fied in the form of INGNepal class. When the clear button is clicked is then it cleared all the in put stored in the form of INGNepal class.

j. **Method Name:** main()

This is the main method to run the java file so, without the main file the program cannot run. So, by the main method it can run in the command prompt(cmd).in the main method of class INGNepal is created which is necessary to call method from the same class.

Evidences

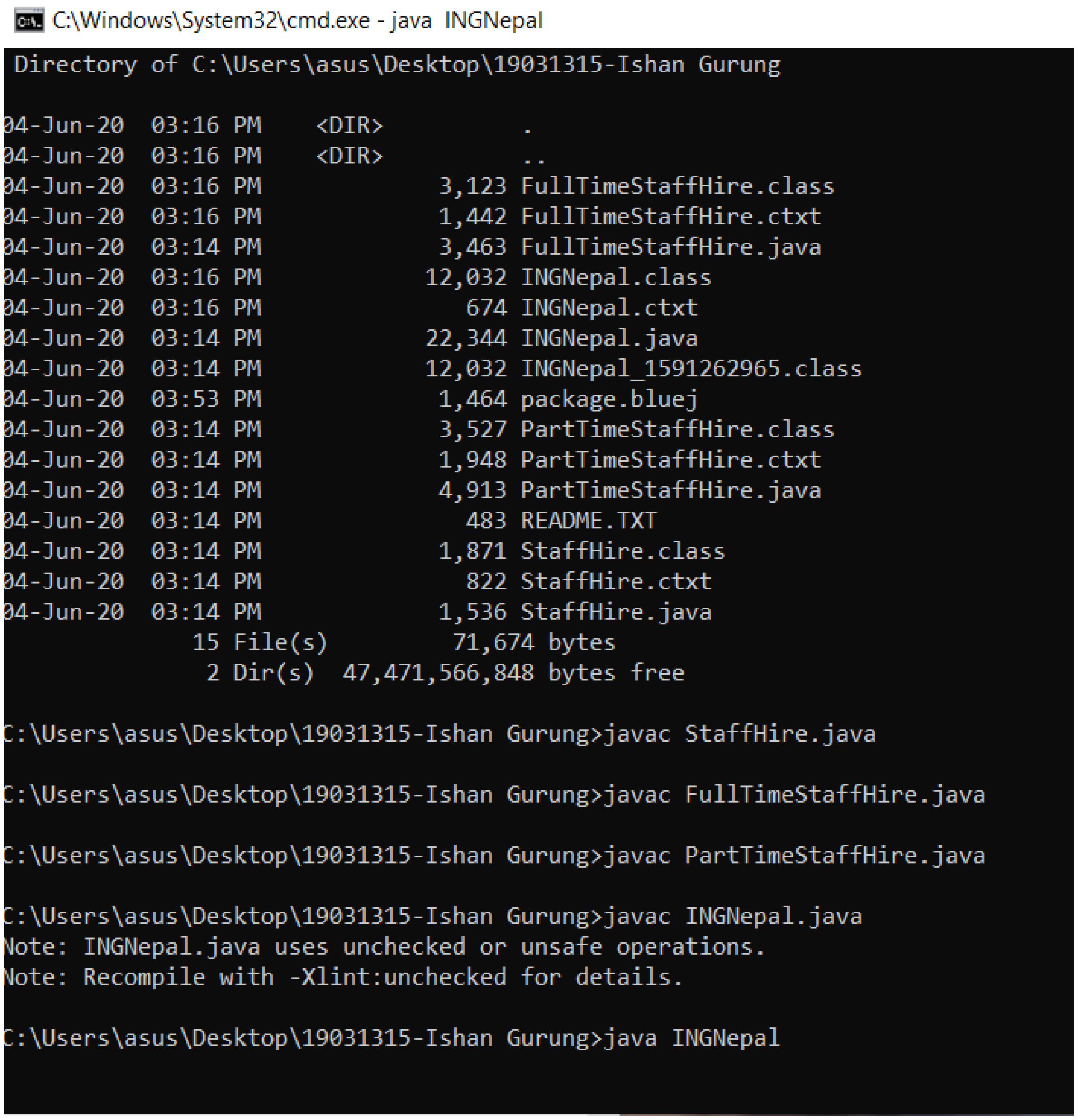
Testing

Test no 1: running program in command promt

Objective	To run INGNepal class in command promt

Action	Instruct command in command prompt to run program.
Expected result	Gui of program should be display
Actual result	It displays.
Conclusion	Test successful

Table 1 test to run GUI from cmd



```
C:\Windows\System32\cmd.exe - java  INGNepal

 Directory of C:\Users\asus\Desktop\19031315-Ishan Gurung

04-Jun-20  03:16 PM    <DIR>          .
04-Jun-20  03:16 PM    <DIR>          ..
04-Jun-20  03:16 PM             3,123 FullTimeStaffHire.class
04-Jun-20  03:16 PM             1,442 FullTimeStaffHire.ctxt
04-Jun-20  03:14 PM             3,463 FullTimeStaffHire.java
04-Jun-20  03:16 PM            12,032 INGNepal.class
04-Jun-20  03:16 PM               674 INGNepal.ctxt
04-Jun-20  03:14 PM            22,344 INGNepal.java
04-Jun-20  03:14 PM            12,032 INGNepal_1591262965.class
04-Jun-20  03:53 PM             1,464 package.bluej
04-Jun-20  03:14 PM             3,527 PartTimeStaffHire.class
04-Jun-20  03:14 PM             1,948 PartTimeStaffHire.ctxt
04-Jun-20  03:14 PM             4,913 PartTimeStaffHire.java
04-Jun-20  03:14 PM               483 README.TXT
04-Jun-20  03:14 PM             1,871 StaffHire.class
04-Jun-20  03:14 PM               822 StaffHire.ctxt
04-Jun-20  03:14 PM             1,536 StaffHire.java
              15 File(s)         71,674 bytes
               2 Dir(s)  47,471,566,848 bytes free

C:\Users\asus\Desktop\19031315-Ishan Gurung>javac StaffHire.java

C:\Users\asus\Desktop\19031315-Ishan Gurung>javac FullTimeStaffHire.java

C:\Users\asus\Desktop\19031315-Ishan Gurung>javac PartTimeStaffHire.java

C:\Users\asus\Desktop\19031315-Ishan Gurung>javac INGNepal.java
Note: INGNepal.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

C:\Users\asus\Desktop\19031315-Ishan Gurung>java INGNepal
```

Figure 1 process in cmd to run in cmd

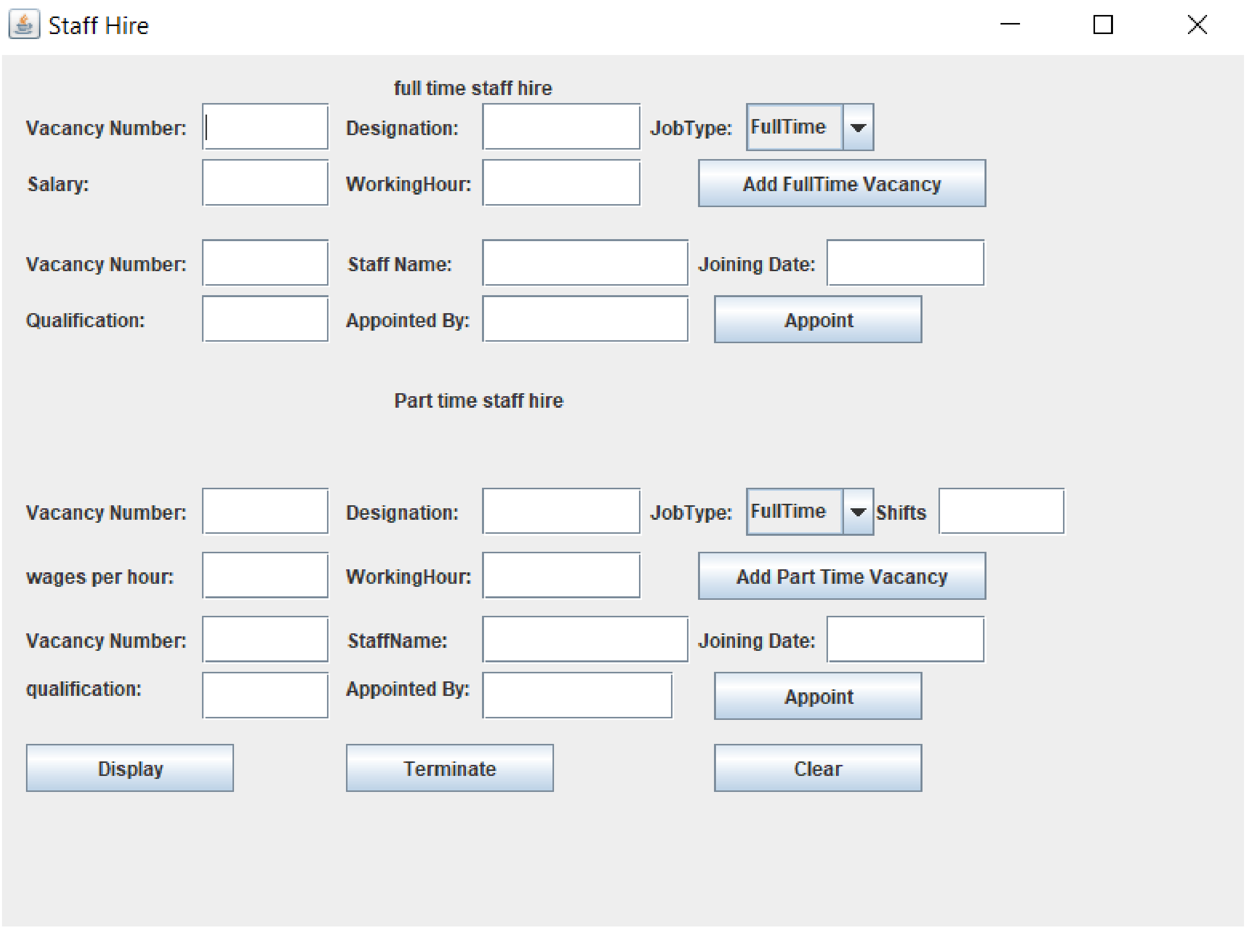


Figure 2 result after the process in cmd

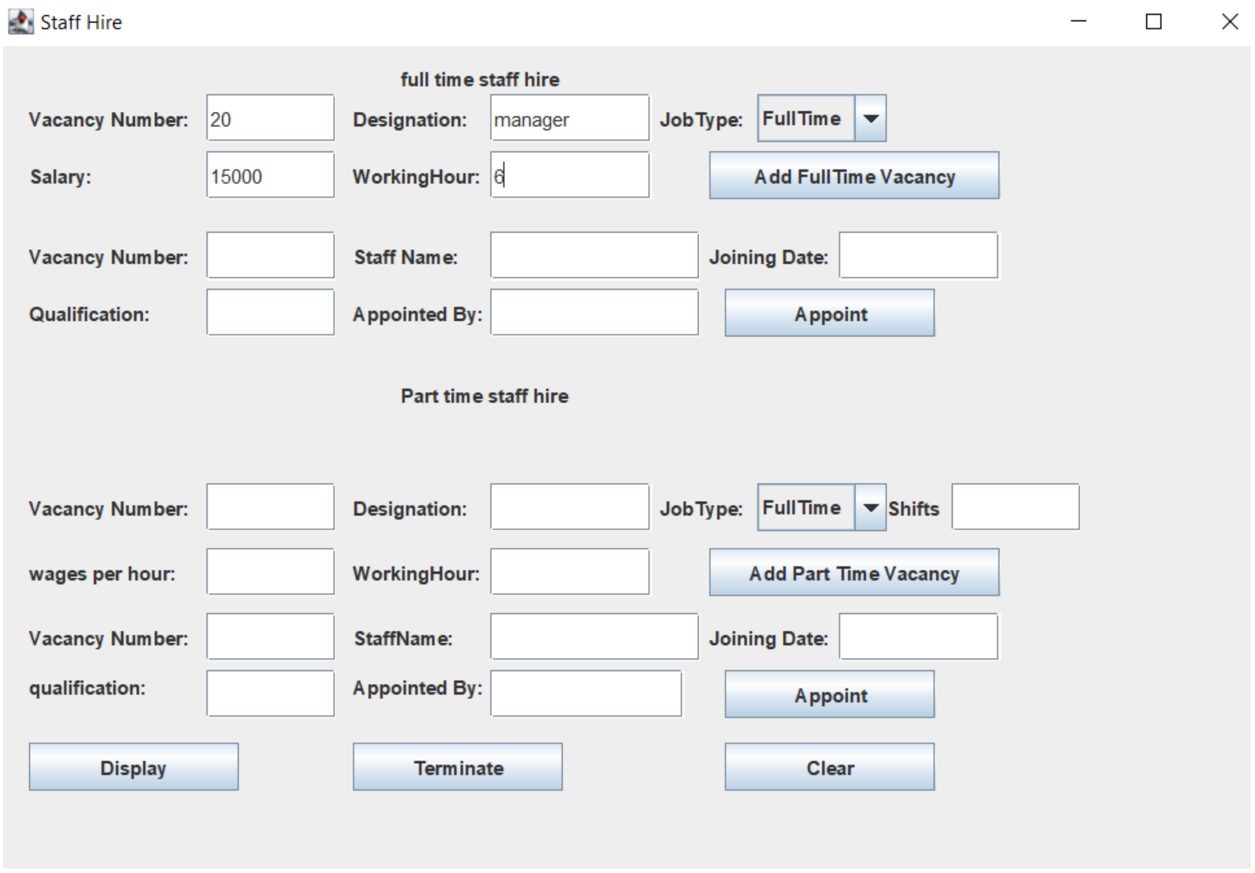
Test no 2:

A add vacancy for Full Time Staff

Test no 2	
Objective	And vacancy text field for fulltime staffhire
Action	Fill-up the text field of fulltime and click to the addvacancy button as Vacancy Number: 20 Designation: “Manager” JobType: “Full Time” Salary: 15000 Working Hour: 6
Expected result	Text should be added and the message should be shown.

Actual result	Staff has added and the message has showed.
conclusion	Test successful

Table 2 adding vacancy in full time staff hire



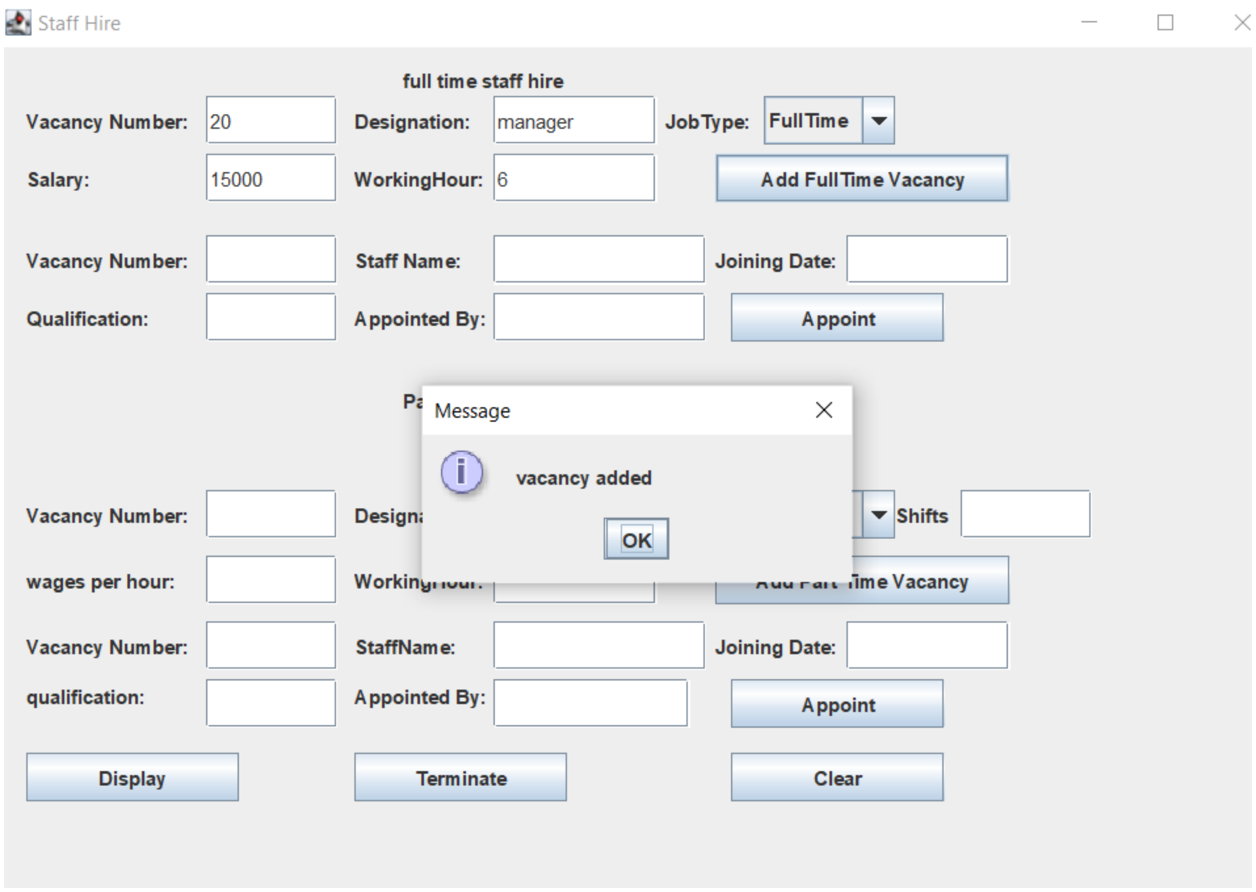
Staff Hire

full time staff hire

Vacancy Number: 20 Designation: manager JobType: FullTime

Salary: 15000 WorkingHour: 6

Figure 3 adding vacancy


Staff Hire

full time staff hire

Vacancy Number: 20
Designation: manager
JobType: FullTime
Salary: 15000
WorkingHour: 6

Message

vacancy added

Figure 4 vacancy finally added

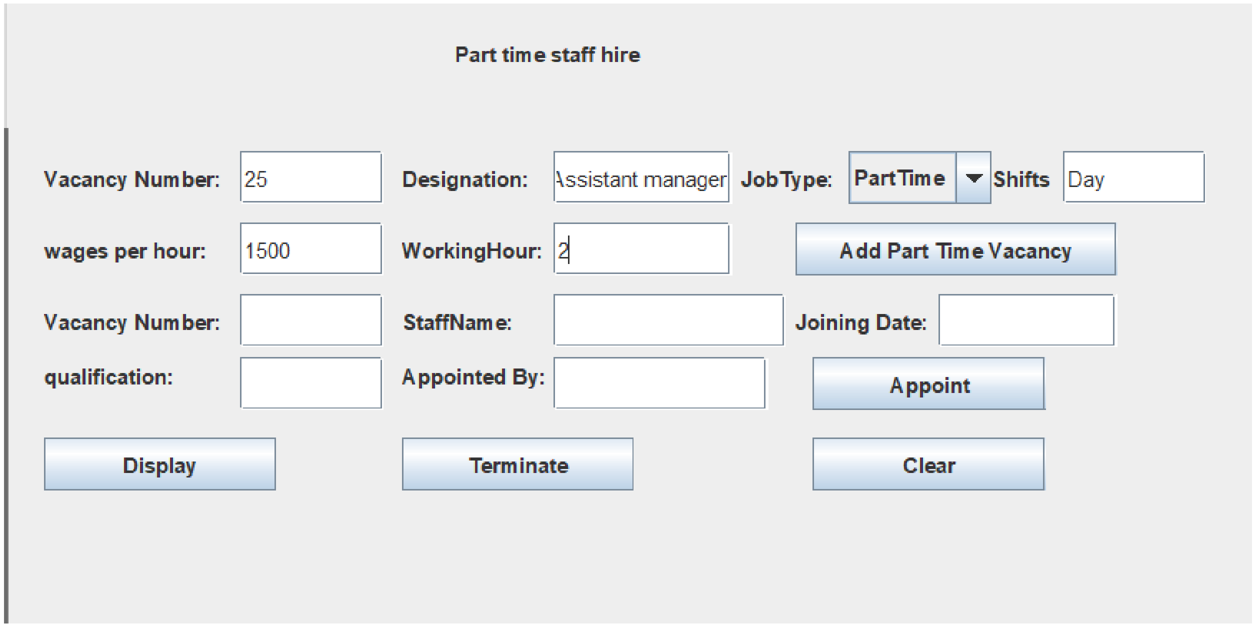
Test no 3:

A add vacancy for partTime Staff

Test no 2	
Objective	And vacancy text field for part-time staffhire
Action	Fill-up the text field of part-time and click to the add vacancy button as Vacancy Number: 25 Designation: “Assistant manager” JobType: “Part Time” Working Hour:2 Wages Per Hour: 1500 Shifts: “Day”
Expected result	Text should be added and the message should be shown.

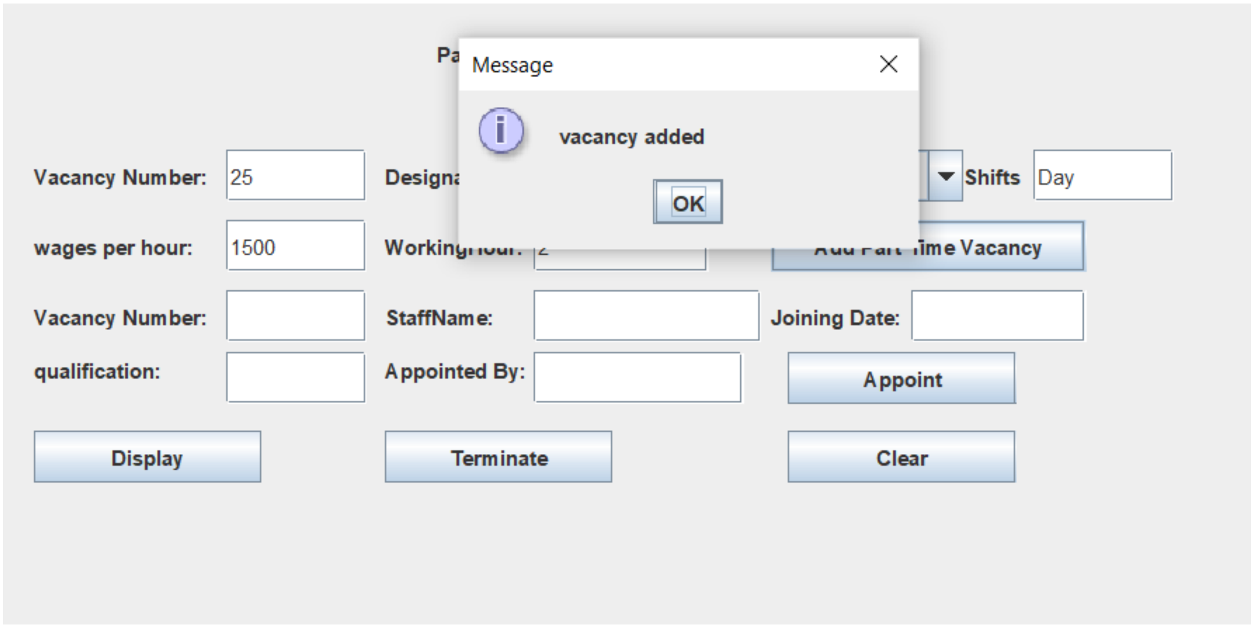
Actual result	Staff has added and the message has showed.
conclusion	Test successful

Table 3 adding vacancy for part time



Vacancy Number: 25
Designation: Assistant manager
JobType: PartTime
Shifts: Day
wages per hour: 1500
WorkingHour: 2

Figure 5process in adding vacancy



vacancy added

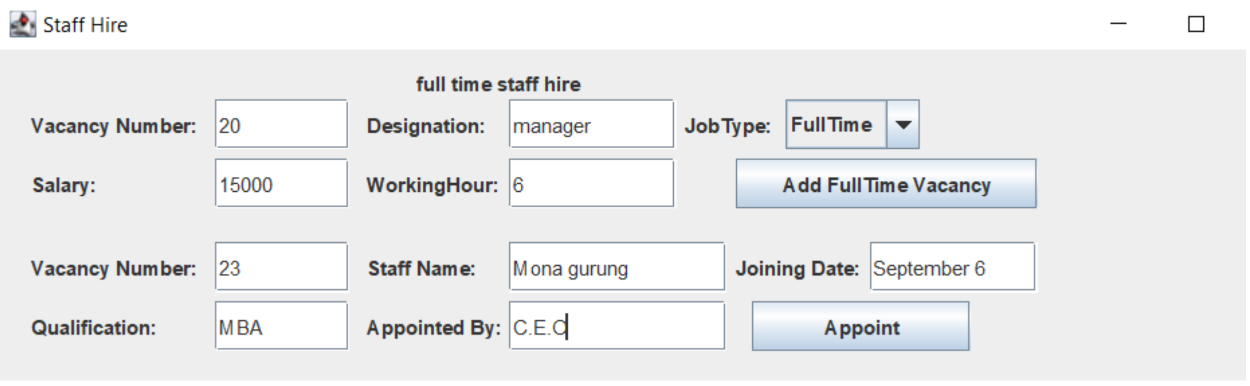
Vacancy Number: 25
Shifts: Day
wages per hour: 1500

Figure 6 finally vacancy added

c. appoint Fulltime staff hire

Objective	To appoint fulltime staff.
Action	Fill-up the text field of fulltime and click to the appoint full time staff hire as Vacancy Number: 23 Staff Name: “Mona gurung” Joining Date:” September 6” Qualification: “MBA” Appointed By:”C.E.O”
Expected result	Staff should be hired and message should be shown
Actual result	Staff has appointed and the message has showed
Conclusion	Test sucessful.

Table 4 adding vacancy for part time staff


Staff Hire

full time staff hire

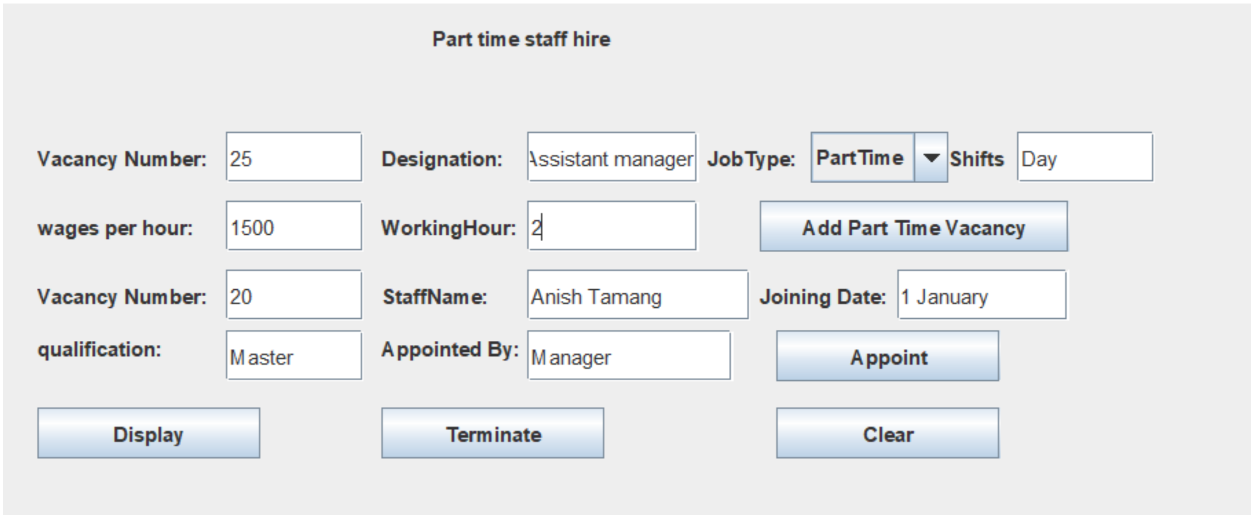
Vacancy Number:	20	Designation:	manager	JobType:	FullTime
Salary:	15000	WorkingHour:	6		
Vacancy Number:	23	Staff Name:	Mona gurung	Joining Date:	September 6
Qualification:	MBA	Appointed By:	C.E.O		

Figure 7 process of adding vacancy in full time

d. Appoint Part time staff hire

Objective	To appoint part-time staff.
Action	Fill-up the text field of part-time and click to the appoint part time staff hire
Expected result	Staff should be hired and message should be shown as Vacancy Number: 20 Staff Name : "Anish Tamang" Joining Date: "1 January" Qualification: "Master" Appointed By: "Manager"
Actual result	Staff has appointed and the message has showed
Conclusion	Test successful.

Table 5adding vacancy for part time



Vacancy Number:	25	Designation:	Assistant manager	JobType:	PartTime	Shifts	Day
wages per hour:	1500	WorkingHour:	2				
Vacancy Number:	20	StaffName:	Anish Tamang	Joining Date:	1 January		
qualification:	Master	Appointed By:	Manager				

Figure 8 imputing value for testing part time

e. Terminate Part Time Staff

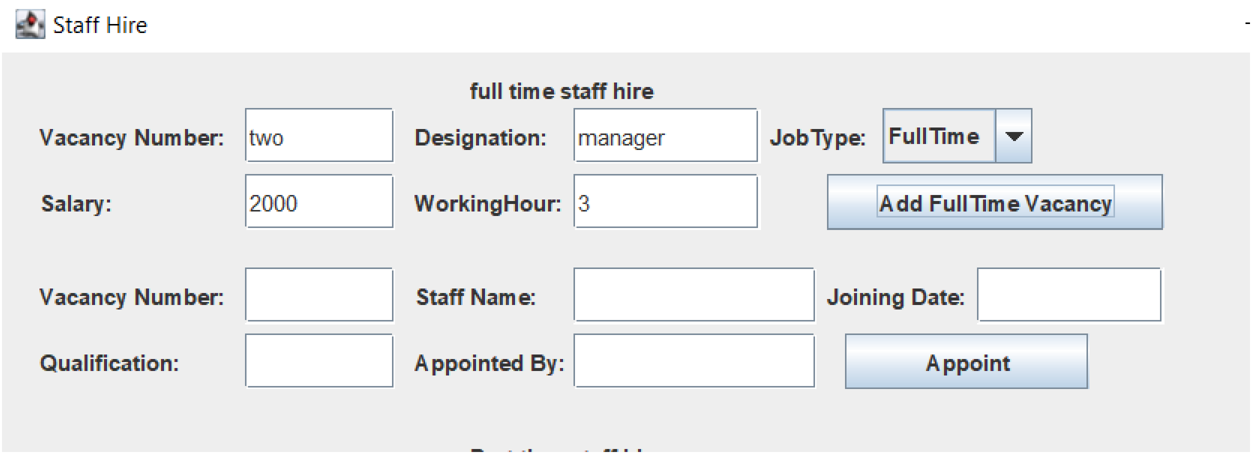
Objective	To terminate part time staff
Action	Fill-up the text field of part-time and click to the terminate part time staff hire
Expected result	Staff should be terminated and message should be shown
Actual result	Staff has terminated and the message has showed
Conclusion	Test successful

Table 6 terminating part time

Test 3 appropriate dialogue box

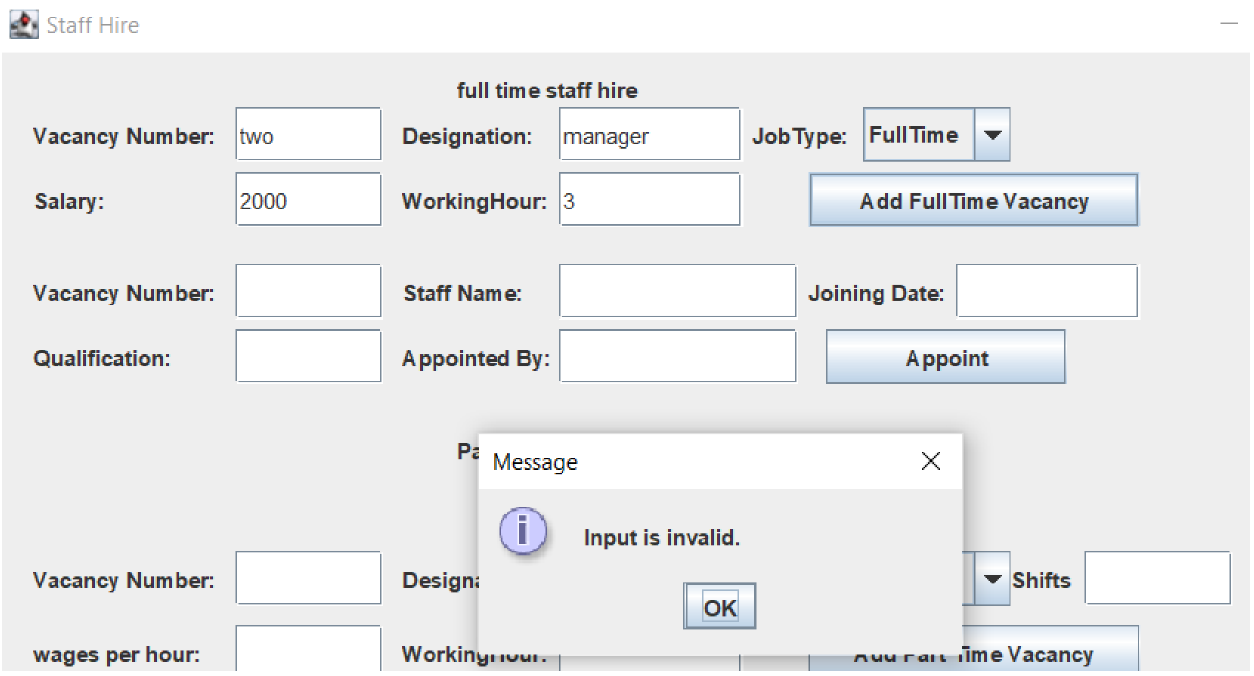
Objective	To enter string character instead of int in vacancy text field.
Action	Vacancy number was filled with string value.
Expected result	Should show message box.
Actual result	Message box has shown
Conclusion	Text successful.

Table 7 testing for the appropriate box



Vacancy Number:	two	Designation:	manager	JobType:	FullTime
Salary:	2000	WorkingHour:	3		
Vacancy Number:		Staff Name:		Joining Date:	
Qualification:		Appointed By:			

Figure 9inputting value for testing



Vacancy Number:	two	Designation:	manager	JobType:	FullTime
Salary:	2000	WorkingHour:	3		
Vacancy Number:		Staff Name:		Joining Date:	
Qualification:		Appointed By:			

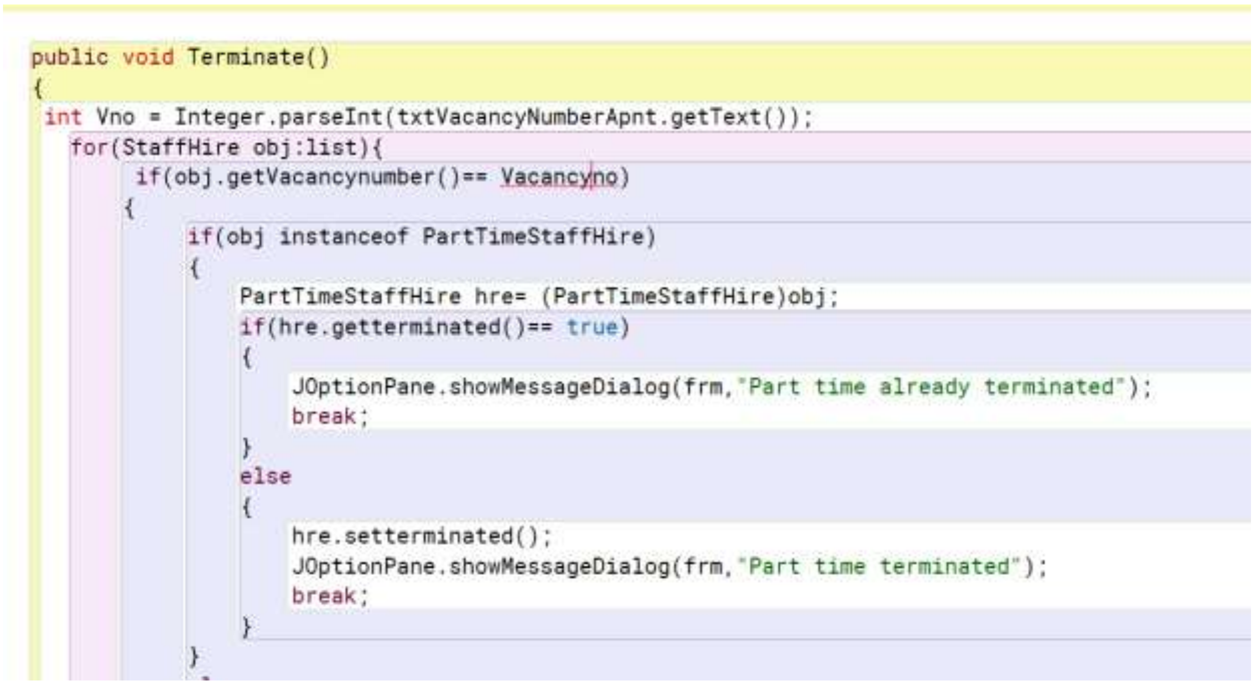
Input is invalid.

Figure 10 result after testing

E. Error Detection and error correction

a. Error Detected

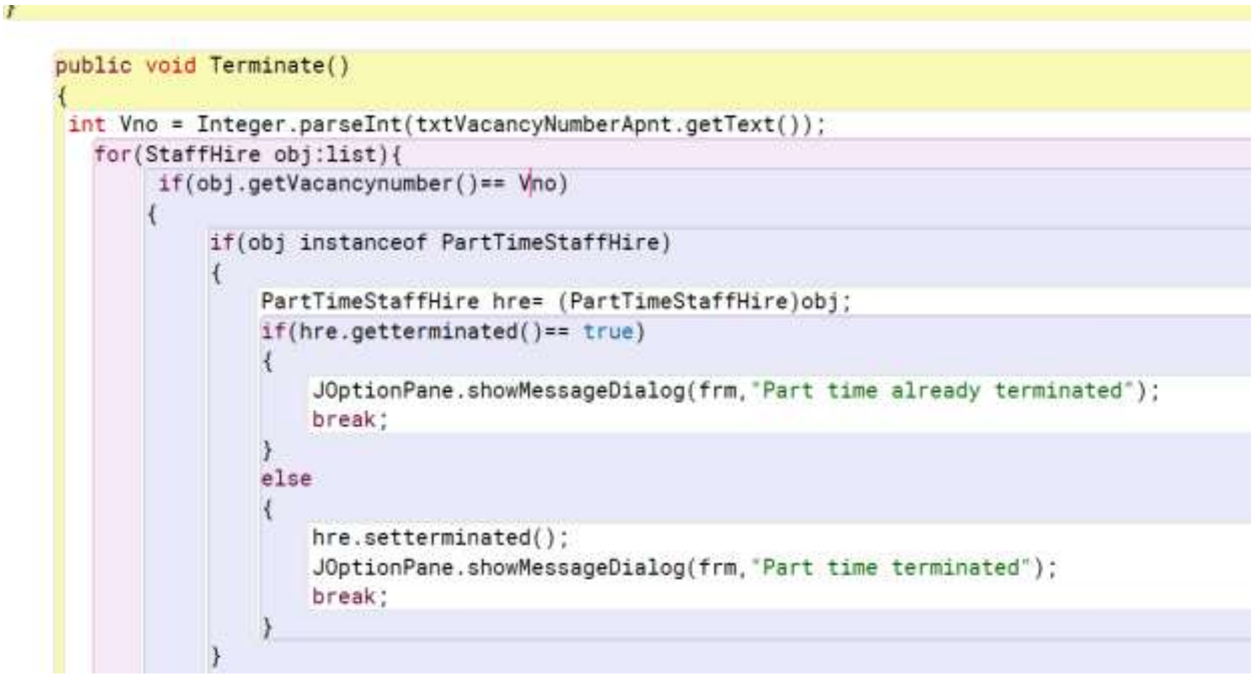
So, the error is syntax error where I have declared variable vacancy number as Vno but I was making mistake instead of writing Vno I has writing Vacancyno so it was showing an error and I have corrected it.



```
public void Terminate()
{
 int Vno = Integer.parseInt(txtVacancyNumberApnt.getText());
   for(StaffHire obj:list){
        if(obj.getVacancynumber()== Vacancyno)
       {
             if(obj instanceof PartTimeStaffHire)
             {
                  PartTimeStaffHire hre= (PartTimeStaffHire)obj;
                  if(hre.getterminated()== true)
                  {
                       JOptionPane.showMessageDialog(frm,"Part time already terminated");
                       break;
                  }
                  else
                  {
                       hre.setterminated();
                       JOptionPane.showMessageDialog(frm,"Part time terminated");
                       break;
                  }
             }
```

Figure 11error 1

Error correction:

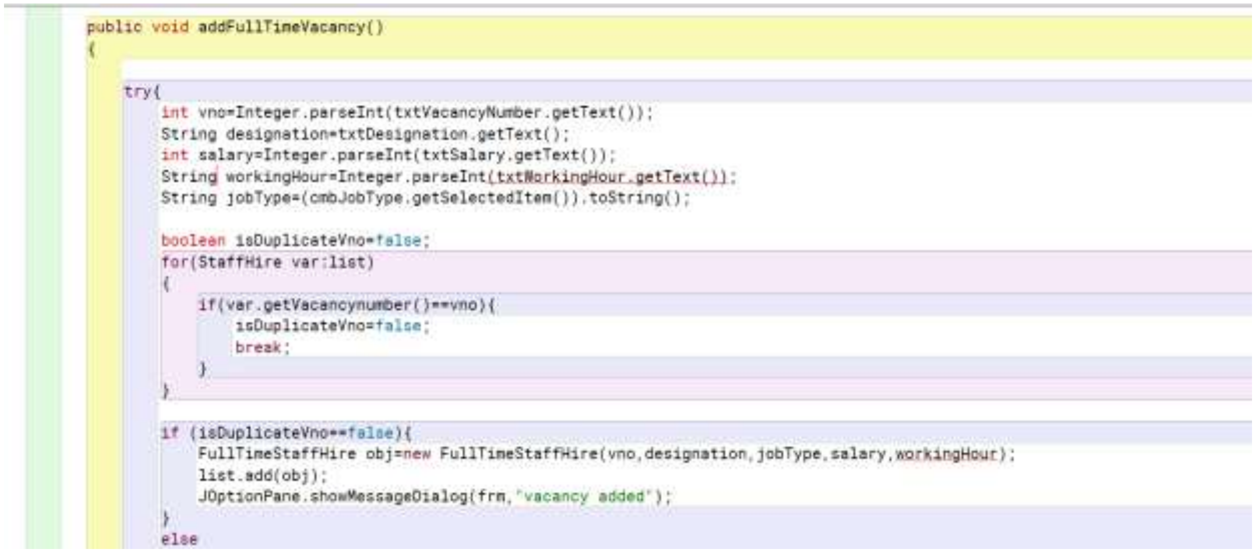


```
public void Terminate()
{
 int Vno = Integer.parseInt(txtVacancyNumberApnt.getText());
   for(StaffHire obj:list){
        if(obj.getVacancynumber()== Vno)
       {
              if(obj instanceof PartTimeStaffHire)
              {
                   PartTimeStaffHire hre= (PartTimeStaffHire)obj;
                   if(hre.getterminated()== true)
                   {
                        JOptionPane.showMessageDialog(frm,"Part time already terminated");
                        break;
                   }
                   else
                   {
                        hre.setterminated();
                        JOptionPane.showMessageDialog(frm,"Part time terminated");
                        break;
                   }
              }
```

Figure 12 syntax error correction

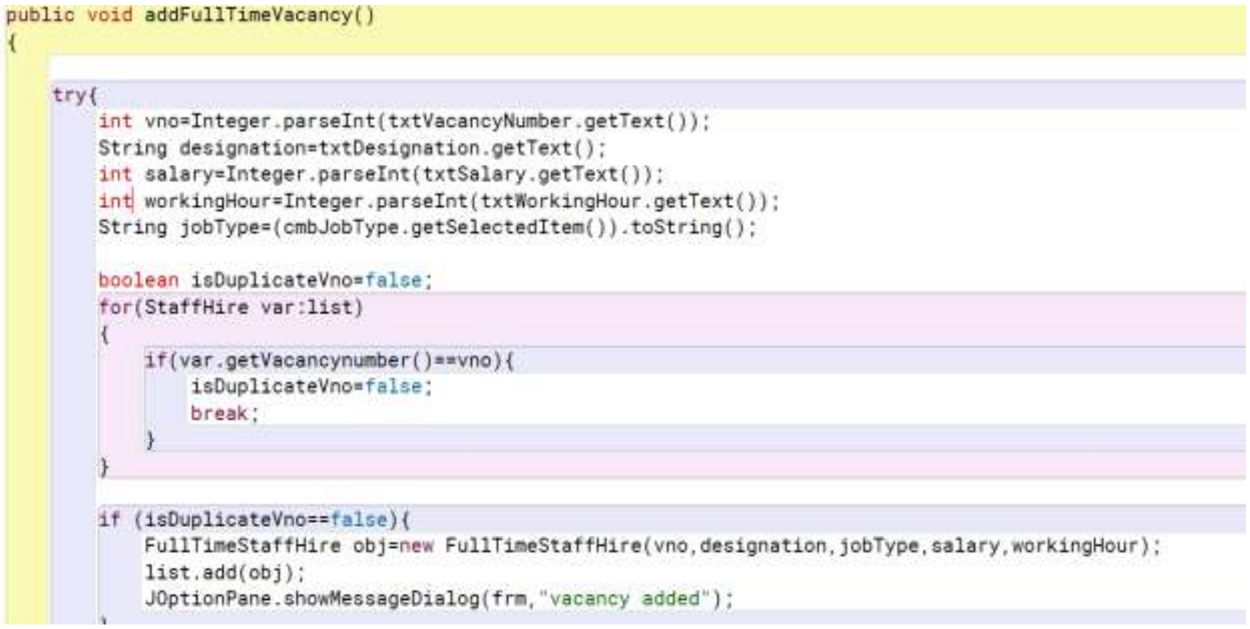
Error detection

Here, I was encounter with another error which is called as logical error. In which I have made mistake was in the working hour instead of putting int in integer type of data type I have assigned it to String type of data type So, I have corrected it and my problem was solved.



```
public void addFullTimeVacancy()
{

    try{
        int vno=Integer.parseInt(txtVacancyNumber.getText());
        String designation=txtDesignation.getText();
        int salary=Integer.parseInt(txtSalary.getText());
        String workingHour=Integer.parseInt(txtWorkingHour.getText());
        String jobType=(cmbJobType.getSelectedItem()).toString();

        boolean isDuplicateVno=false;
        for(StaffHire var:list)
        {
            if(var.getVacancynumber()==vno){
                isDuplicateVno=false;
                break;
            }
        }

        if (isDuplicateVno==false){
            FullTimeStaffHire obj=new FullTimeStaffHire(vno,designation,jobType,salary,workingHour);
            list.add(obj);
            JOptionPane.showMessageDialog(frm,"vacancy added");
        }
        else
```

Figure 13 logical error

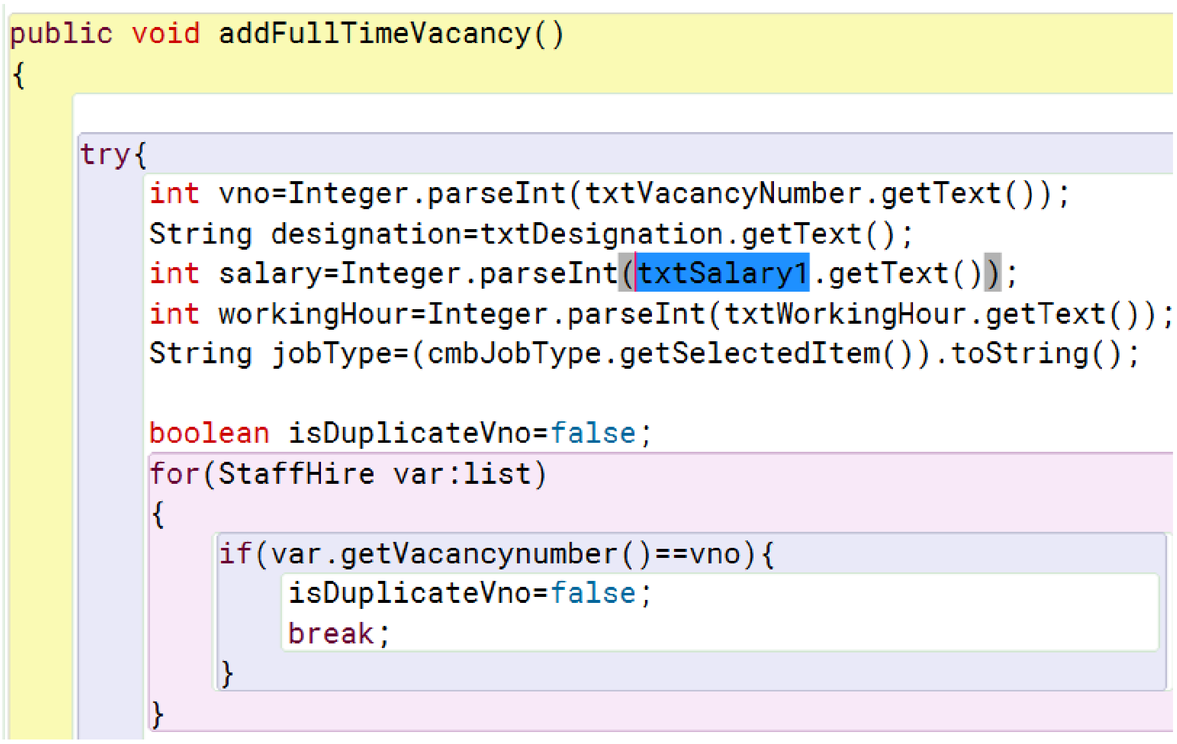


```
public void addFullTimeVacancy()
{

    try{
        int vno=Integer.parseInt(txtVacancyNumber.getText());
        String designation=txtDesignation.getText();
        int salary=Integer.parseInt(txtSalary.getText());
        int workingHour=Integer.parseInt(txtWorkingHour.getText());
        String jobType=(cmbJobType.getSelectedItem()).toString();

        boolean isDuplicateVno=false;
        for(StaffHire var:list)
        {
            if(var.getVacancynumber()==vno){
                isDuplicateVno=false;
                break;
            }
        }

        if (isDuplicateVno==false){
            FullTimeStaffHire obj=new FullTimeStaffHire(vno,designation,jobType,salary,workingHour);
            list.add(obj);
            JOptionPane.showMessageDialog(frm,"vacancy added");
```

Figure 14logical error correction

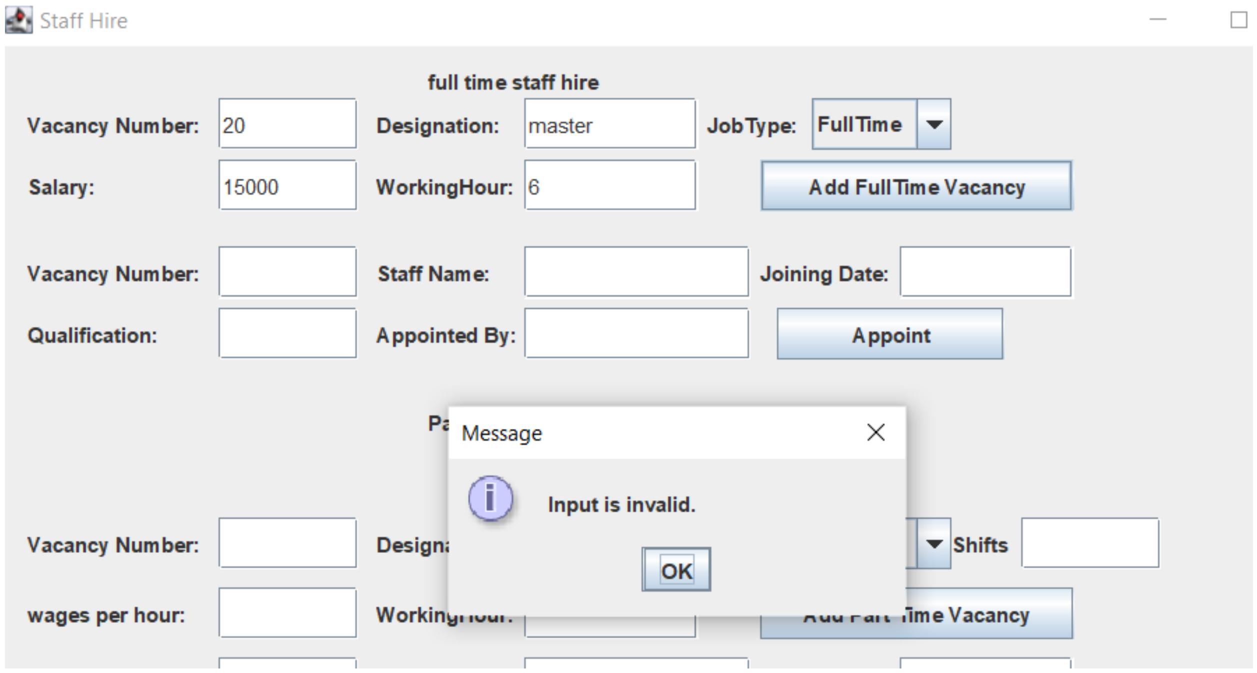
Test no 3

Here is the error which runs the code but while adding the vacancy the vacancy the result comes as input invalid so I have encountered as the variable incorrect and the result is below so I have corrected it.



```
public void addFullTimeVacancy()
{

    try{
        int vno=Integer.parseInt(txtVacancyNumber.getText());
        String designation=txtDesignation.getText();
        int salary=Integer.parseInt(txtSalary1.getText());
        int workingHour=Integer.parseInt(txtWorkingHour.getText());
        String jobType=(cmbJobType.getSelectedItem()).toString();

        boolean isDuplicateVno=false;
        for(StaffHire var:list)
        {
            if(var.getVacancynumber()==vno){
                isDuplicateVno=false;
                break;
            }
        }
```

Figure 15 error 3

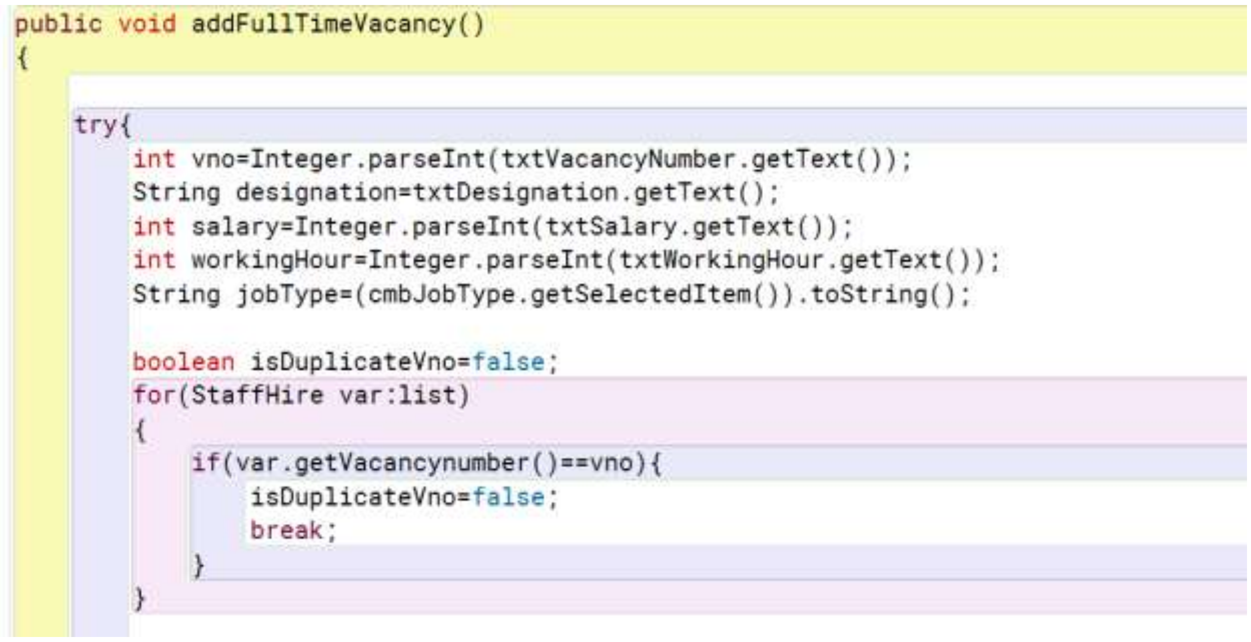


Vacancy Number:	20	Designation:	master	JobType:	FullTime
Salary:	15000	WorkingHour:	6		

Message

Input is invalid.

Figure 16 error 3 in vacancy frame



```
public void addFullTimeVacancy()
{

    try{
        int vno=Integer.parseInt(txtVacancyNumber.getText());
        String designation=txtDesignation.getText();
        int salary=Integer.parseInt(txtSalary.getText());
        int workingHour=Integer.parseInt(txtWorkingHour.getText());
        String jobType=(cmbJobType.getSelectedItem()).toString();

        boolean isDuplicateVno=false;
        for(StaffHire var:list)
        {
            if(var.getVacancynumber()==vno){
                isDuplicateVno=false;
                break;
            }
        }
```

Figure 17 error correction

Conclusion

Doing this coursework was very tough for me personally. It was not possible to build whole task in a single try or a single day. to complete every single question from this coursework was high from my comfort level. At first I was even not able to understand the question properly when it was published but when I started to read the coursework and discussed with the teacher and my fellow friends then slowly it start to make sense. Firstly I researched about the whole coursework and by knowing the question I started to make the GUI of INGNepal .

Firstly, to make the GUI of INGNepal class was very difficult for me personally. But after I seen the sample of GUI from the google classroom page, and from ther I get proper knowledge about GUI. There was some problem with the code so with the help of tutor teacher and my friends I tackle all these problems. After many errors I have finally completed the function of making GUI of staffhiring as mentioned in our coursework .

From this coursework, I am able to make the application for teacher to hire in school, college, institute basically it helps to make the application to hire the staff in any organization. After the completion of all the coding part thoroughly moved, I moved toward test for different coding. I run the various methods by inspecting and re-inspecting it .

While completing the whole coursework it was very interesting for me in developing GUI and to run the button.

Appendix 1

```
/**
 * Write a description of class INGNepal here.
 *
 * @author (your name)Ishan Gurung
 * @version (a version number or a date)
 */

import javax.swing.*;
import java.awt.event.*;
import java.util.*;




public class INGNepal implements ActionListener
{
    JFrame frm;

    JLabel title,title1,lblVacancyNumber,
lblDesignation,lblJobType,
lblSalary,lblWorkingHour,lblVacancyNumberApnt,
lblStaffName,lblJoinDate,lblQualification,lblApointedBy,lblVacancy
Number1, lblDesignaion1,lblJobType1, lblSalary1,

lblWorkingHour1,lblVacancyNumberApnt1,lblshifts,lblStaffName1,
lblJoinDate1,lblQualification1,lblApointedBy1;


        JTextField txtVacancyNumber,
txtDesignation,txtJobType,
txtSalary,txtWorkingHour,txtVacancyNumberApnt,

txtStaffName,txtJoinDate,txtQualification,txtApointedBy,txtVacanc
yNumber1, txtDesignation1,txtJobTyp1e, txtSalary1,
          txtWorkingHour1,txtVacancyNumberApnt1,

txtStaffName1,txtJoinDate1,txtQualification1,txtApointedBy1,txtshi
fts;


        JComboBox cmbJobType,cmbJobType1;
        JButton btnAddFullTimeVacancy,
btnAddPartTimeVacancy,
btnAppointFT,btnAppointFT1,btnDisplay,btnClear,btnTerminate;
        String VacancyNumber;
        ArrayList<StaffHire> list=new ArrayList<StaffHire>();

        public void StaffHireForm()
        {
```

```
frm=new JFrame("Staff Hire");
frm.setSize(800,600);
frm.setLayout(null);



title1=new JLabel("full time staff hire");
title1.setBounds(250,5,130,30);
frm.add(title1);

lblVacancyNumber=new JLabel("Vacancy Number:");
lblVacancyNumber.setBounds(20,30,120,30);
frm.add(lblVacancyNumber);

txtVacancyNumber=new JTextField();
txtVacancyNumber.setBounds(130,30,80,30);
frm.add(txtVacancyNumber);

lblDesignation=new JLabel("Designation:");
lblDesignation.setBounds(220,30,130,30);
frm.add(lblDesignation);

txtDesignation=new JTextField();
txtDesignation.setBounds(305,30,100,30);
```

```
frm.add(txtDesignation);

lblJobType=new JLabel("JobType:");
lblJobType.setBounds(410,30,80,30);
frm.add(lblJobType);

String jobType[]={"FullTime","PartTime"};
cmbJobType=new JComboBox(jobType);
cmbJobType.setBounds(470,30,80,30);
frm.add(cmbJobType);

lblSalary=new JLabel("Salary:");
lblSalary.setBounds(20,65,120,30);
frm.add(lblSalary);

txtSalary=new JTextField();
txtSalary.setBounds(130,65,80,30);
frm.add(txtSalary);

lblWorkingHour=new JLabel("WorkingHour:");
lblWorkingHour.setBounds(220,65,100,30);
frm.add(lblWorkingHour);
```

```
                txtWorkingHour=new JTextField();
                txtWorkingHour.setBounds(305,65,100,30);
                frm.add(txtWorkingHour);

                btnAddFullTimeVacancy=new JButton("Add FullTime
Vacancy");
                btnAddFullTimeVacancy.setBounds(440,65,180,30);
                frm.add(btnAddFullTimeVacancy);
                btnAddFullTimeVacancy.addActionListener(this);

                lblVacancyNumberApnt=new JLabel("Vacancy
Number:");
                lblVacancyNumberApnt.setBounds(20,115,120,30);
                frm.add(lblVacancyNumberApnt);

                txtVacancyNumberApnt=new JTextField();
                txtVacancyNumberApnt.setBounds(130,115,80,30);
                frm.add(txtVacancyNumberApnt);

                lblStaffName=new JLabel("Staff Name:");
                lblStaffName.setBounds(220,115,100,30);
                frm.add(lblStaffName);

                txtStaffName=new JTextField();
```

```
txtStaffName.setBounds(305,115,130,30);
frm.add(txtStaffName);

lblJoinDate=new JLabel("Joining Date:");
lblJoinDate.setBounds(440,115,100,30);
frm.add(lblJoinDate);

txtJoinDate=new JTextField();
txtJoinDate.setBounds(520,115,100,30);
frm.add(txtJoinDate);

lblQualification=new JLabel("Qualification:");
lblQualification.setBounds(20,150,120,30);
frm.add(lblQualification);

txtQualification=new JTextField();
txtQualification.setBounds(130,150,80,30);
frm.add(txtQualification);

lblApointedBy=new JLabel("Appointed By:");
lblApointedBy.setBounds(220,150,100,30);
frm.add(lblApointedBy);
```

```
        txtApointedBy=new JTextField();
        txtApointedBy.setBounds(305,150,130,30);
        frm.add(txtApointedBy);

        btnAppointFT=new JButton("Appoint");
        btnAppointFT.setBounds(450,150,130,30);
        frm.add(btnAppointFT);
        btnAppointFT.addActionListener(this);



        title=new JLabel("Part time staff hire");
        title.setBounds(250,200,130,30);
        frm.add(title);

        lblVacancyNumber1=new JLabel("Vacancy
Number:");
        lblVacancyNumber1.setBounds(20,270,120,30);
        frm.add(lblVacancyNumber1);

        txtVacancyNumber1=new JTextField();
        txtVacancyNumber1.setBounds(130,270,80,30);
        frm.add(txtVacancyNumber1);
```

```
lblDesignaion1=new JLabel("Designation:");
lblDesignaion1.setBounds(220,270,130,30);
frm.add(lblDesignaion1);

txtDesignation1=new JTextField();
txtDesignation1.setBounds(305,270,100,30);
frm.add(txtDesignation1);

lblJobType1=new JLabel("JobType:");
lblJobType1.setBounds(410,270,80,30);
frm.add(lblJobType1);

String jobType1[]={"FullTime","PartTime"};
cmbJobType1=new JComboBox(jobType);
cmbJobType1.setBounds(470,270,80,30);
frm.add(cmbJobType1);

lblshifts=new JLabel("Shifts");
lblshifts.setBounds(550,270,80,30);
frm.add(lblshifts);

txtshifts =new JTextField();
txtshifts.setBounds(590,270,80,30);
```

```
                frm.add(txtshifts);



                lblSalary1=new JLabel("wages per hour:");
                lblSalary1.setBounds(20,310,120,30);
                frm.add(lblSalary1);

                txtSalary1=new JTextField();
                txtSalary1.setBounds(130,310,80,30);
                frm.add(txtSalary1);

                lblWorkingHour1=new JLabel("WorkingHour:");
                lblWorkingHour1.setBounds(220,310,100,30);
                frm.add(lblWorkingHour1);

                txtWorkingHour1=new JTextField();
                txtWorkingHour1.setBounds(305,310,100,30);
                frm.add(txtWorkingHour1);

                btnAddPartTimeVacancy=new JButton("Add Part
Time Vacancy");

btnAddPartTimeVacancy.setBounds(440,310,180,30);
                frm.add(btnAddPartTimeVacancy);
```

```
        btnAddPartTimeVacancy.addActionListener(this);

        lblVacancyNumberApnt1=new JLabel("Vacancy
Number:");
        lblVacancyNumberApnt1.setBounds(20,350,120,30);
        frm.add(lblVacancyNumberApnt1);

        txtVacancyNumberApnt1=new JTextField();
        txtVacancyNumberApnt1.setBounds(130,350,80,30);
        frm.add(txtVacancyNumberApnt1);

        lblStaffName1=new JLabel("StaffName:");
        lblStaffName1.setBounds(220,350,100,30);
        frm.add(lblStaffName1);

        txtStaffName1=new JTextField();
        txtStaffName1.setBounds(305,350,130,30);
        frm.add(txtStaffName1);

        lblJoinDate1=new JLabel("Joining Date:");
        lblJoinDate1.setBounds(440,350,100,30);
        frm.add(lblJoinDate1);
```

```
txtJoinDate1=new JTextField();
txtJoinDate1.setBounds(520,350,100,30);
frm.add(txtJoinDate1);

lblQualification1=new JLabel("qualification:");
lblQualification1.setBounds(20,380,120,30);
frm.add(lblQualification1);

txtQualification1=new JTextField();
txtQualification1.setBounds(130,385,80,30);
frm.add(txtQualification1);

lblApointedBy1=new JLabel("Appointed By:");
lblApointedBy1.setBounds(220,380,100,30);
frm.add(lblApointedBy1);

txtApointedBy1=new JTextField();
txtApointedBy1.setBounds(305,385,120,30);
frm.add(txtApointedBy1);

btnAppointFT1=new JButton("Appoint");
btnAppointFT1.setBounds(450,385,130,30);
frm.add(btnAppointFT1);
```

```
        btnAppointFT1.addActionListener(this);

        btnDisplay=new JButton("Display");
        btnDisplay.setBounds(20,430,130,30);
        frm.add(btnDisplay);
        btnDisplay.addActionListener(this);

        btnTerminate =new JButton("Terminate");
        btnTerminate.setBounds(220,430,130,30);
        frm.add(btnTerminate);
        btnTerminate.addActionListener(this);

        btnClear =new JButton("Clear");
        btnClear.setBounds(450,430,130,30);
        frm.add(btnClear);
        btnClear.addActionListener(this);



        frm.setVisible(true);

frm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
```

```
   public void actionPerformed(ActionEvent e)
{
      if (e.getSource()==btnAddFullTimeVacancy)
    {
       addFullTimeVacancy();
    }
    if (e.getSource()==btnAddPartTimeVacancy)
    {
       addPartTimeVacancy();
    }
     else if(e.getSource()==btnAppointFT)
     {
        appointFullTime();
     }
     else if(e.getSource()==btnAppointFT1)
     {
        appointPartTime();
     }
     else if(e.getSource()==btnDisplay)
     {
        Display();
     }
```

```
            else if(e.getSource()==btnTerminate)
            {
                Terminate();
            }
            else if (e.getSource()==btnClear)
            {
                Clear();
            }

        }



        public void addFullTimeVacancy()
        {

            try{
                int
vno=Integer.parseInt(txtVacancyNumber.getText());
                String designation=txtDesignation.getText();
                int salary=Integer.parseInt(txtSalary.getText());
                int
workingHour=Integer.parseInt(txtWorkingHour.getText());
```

```
            String
jobType=(cmbJobType.getSelectedItem()).toString();


            boolean isDuplicateVno=false;
            for(StaffHire var:list)
            {
              if(var.getVacancynumber()==vno){
                isDuplicateVno=false;
                break;
              }
            }


            if (isDuplicateVno==false){
              FullTimeStaffHire obj=new
FullTimeStaffHire(vno,designation,jobType,salary,workingHour);
              list.add(obj);
              JOptionPane.showMessageDialog(frm,"vacancy
added");
            }
            else
            {
              JOptionPane.showMessageDialog(frm,"Input
Vacancy no is already in the list."+list.size());
            }
```

```
        }
        catch(Exception exp)
        {
            JOptionPane.showMessageDialog(frm,"Input is
invalid.");
        }

    }

     public void appointFullTime()
     {

        int
vno=Integer.parseInt(txtVacancyNumberApnt.getText());
        String staffName=txtStaffName.getText();
        String joinDate=txtJoinDate.getText();
        String qualification=txtQualification.getText();
        String appointedBy=txtApointedBy.getText();

        boolean vnoFound=false;
        for(StaffHire obj:list){
            if(obj.getVacancynumber()==vno){
                vnoFound=true;
                if(obj instanceof FullTimeStaffHire){
                    FullTimeStaffHire h1=(FullTimeStaffHire)obj;
                    if(h1.getjoined()==true){
                        JOptionPane.showMessageDialog(frm,"Staff
already hired!");
                    }else{

h1.hirefulltimestaff(staffName,joinDate,qualification,appointedBy);
                        JOptionPane.showMessageDialog(frm,"Staff
has been hired!");
                        break;
                    }
                }else{
                    JOptionPane.showMessageDialog(frm,"It is not
for Parttime staff Hire");
                    break;
            }

if(!vnoFound){JOptionPane.showMessageDialog(frm,"invalid
vacancy");
            }
        }
    }
  }

    public void Display()
```

```
        {
            for(StaffHire obj:list)
            {
                if(obj instanceof FullTimeStaffHire)
                {
                    obj=(FullTimeStaffHire)obj;
                    obj.display();
                }
                else if(obj instanceof PartTimeStaffHire)
                {
                    PartTimeStaffHire obj11 =(PartTimeStaffHire)obj;
                    obj11.display();
                }
            }

        }


            public void Terminate()
            {
             int Vno =
Integer.parseInt(txtVacancyNumberApnt.getText());
               for(StaffHire obj:list){
                    if(obj.getVacancynumber()== Vno)
                    {
                        if(obj instanceof PartTimeStaffHire)
                        {
                            PartTimeStaffHire hre=
(PartTimeStaffHire)obj;
                            if(hre.getterminated()== true)
                            {

JOptionPane.showMessageDialog(frm,"Part time already
terminated");
                                break;
                            }
                            else
                            {
                                hre.setterminated();

JOptionPane.showMessageDialog(frm,"Part time terminated");
                                break;
                            }
                        }
                        else
                        {
                            JOptionPane.showMessageDialog(frm,"Not
for part time");
                            break;
                        }


                    }
                    else
                    {
                        JOptionPane.showMessageDialog(frm,"Enter
vacancy Number");
                    }
                }
            }

            public void Clear()
            {
            txtVacancyNumber.setText(" ");
            txtDesignation.setText(" ");
            cmbJobType.setSelectedIndex(0);
            txtSalary.setText(" ");
            txtWorkingHour.setText(" ");



            txtVacancyNumberApnt.setText(" ");
            txtStaffName.setText(" ");
            txtJoinDate.setText(" ");
```

```
            txtQualification.setText(" ");
            txtApointedBy.setText(" ");



            txtVacancyNumber1.setText(" ");
            txtDesignation1.setText(" ");
            cmbJobType1.setSelectedIndex(0);
            txtshifts.setText(" ");
            txtSalary1.setText(" ");
            txtWorkingHour1.setText(" ");


            txtVacancyNumberApnt1.setText(" ");
            txtStaffName1.setText(" ");
            txtJoinDate1.setText(" ");
            txtQualification1.setText(" ");
            txtApointedBy1.setText(" ");


           JOptionPane.showMessageDialog(frm,"Text fields
cleared");
        }
```

```
     public void addPartTimeVacancy()
    {

      try{
         int
vacancynumber=Integer.parseInt(txtVacancyNumber1.getText());
         String designation=txtDesignation1.getText();
         int
wagesperhour=Integer.parseInt(txtSalary1.getText());
         int
workingHour=Integer.parseInt(txtWorkingHour1.getText());
         String
jobType=(cmbJobType1.getSelectedItem()).toString();
         String shifts=txtshifts.getText();
         boolean isDuplicateVno=false;
         for(StaffHire var:list)
         {
           if(var.getVacancynumber()==vacancynumber){
             isDuplicateVno=false;
             break;
           }
         }
```

```
                if (isDuplicateVno==false){
                    PartTimeStaffHire obj=new PartTimeStaffHire
(vacancynumber,designation,jobType,wagesperhour,workingHour
,shifts);
                    list.add(obj);
                    JOptionPane.showMessageDialog(frm,"vacancy
added");
                }
                else
                {
                    JOptionPane.showMessageDialog(frm,"Input
Vacancy no is already in the list.");
                }
            }
            catch(Exception exp)
            {
                JOptionPane.showMessageDialog(frm,"Input is
invalid.");
            }


        }

         public void appointPartTime()
         {
```

```
        int
vacancynumber=Integer.parseInt(txtVacancyNumberApnt.getText
());
        String staffName1=txtStaffName.getText();
        String joiningDate1=txtJoinDate.getText();
        String qualification1=txtQualification.getText();
        String appointedBy1=txtApointedBy.getText();

        boolean vnoFounds=false;
        for(StaffHire obj:list){
          if(obj.getVacancynumber()==vacancynumber){
            vnoFounds=true;
            if(obj instanceof PartTimeStaffHire){
              PartTimeStaffHire h2=(PartTimeStaffHire)obj;
              if(h2.getjoined()==true){
                JOptionPane.showMessageDialog(frm,"Staff
already hired!");
                break;
              }else{
                h2.
hireparttimestaffhire(staffName1,joiningDate1,qualification1,appoi
ntedBy1);
                JOptionPane.showMessageDialog(frm,"Staff
has been hired!");
```

```
                    break;
                  }
                }else{
                  JOptionPane.showMessageDialog(frm,"It is not for
Parttime staff Hire");
                  break;
                }
              }
              else{JOptionPane.showMessageDialog(frm,"invalid
vacancy");
              }
            }
          }



    public static void main(String[]args)
    {
      INGNepal A=new INGNepal();
      A.StaffHireForm();
    }
  }
```

Appendix2

Introduction

Working on any kind of project or coursework is interesting as well as challenging too. 11 Dec 2019, our coursework was released, which is all about implementing the concept of classes and objects. In this coursework, we have to write code for three different classes, which are to be linked with each other using the concept of Inheritance. They are StaffHire, FullTimeStaffHire and PartTimeStaffHire. Among them, StaffHire is the parent class or base class whereas FullTimeSatffHire and PartTimeSatffHire are child class or derived class. This coursework helps us to hire staff according to requirements , to test the details of the workforce and the fame of the field workers. In other words, the main objective of this coursework is to learn about the concept of object-oriented Programming (OOP) and implement it in developing real-world system namely Developer Appointing System.

Class Diagram

Class Diagram is the tabular representation of all the methods and variables that we have used while writing code for any particular class. It consists of three different blocks, which holds class name, instance variables and methods in a sequential order. Likewise, there is sign convention for writing class diagram. We need to use them according to access modifier that we have used with instance variables and methods. For example, we need to use "+" sign for public access modifier and "–" sign for private access modifier. Likewise, there are other conventions too.

Class diagram for each of the class are as follows:

Class Diagram

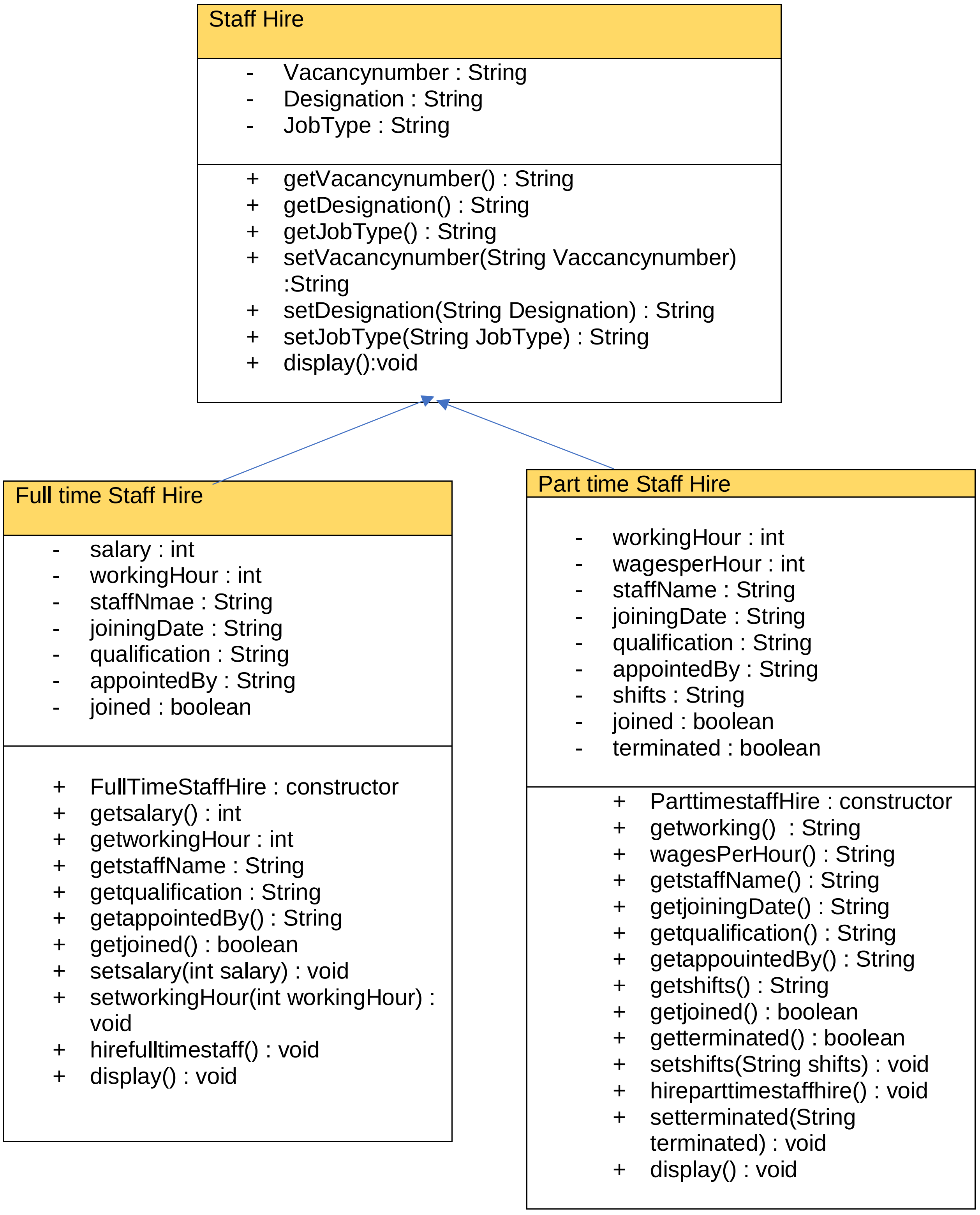

Staff Hire
- Vacancynumber : String
- Designation : String
- JobType : String
+ getVacancynumber() : String
+ getDesignation() : String
+ getJobType() : String
+ setVacancynumber(String Vaccancynumber) :String
+ setDesignation(String Designation) : String
+ setJobType(String JobType) : String
+ display():void
Full time Staff Hire
- salary : int
- workingHour : int
- staffNmae : String
- joiningDate : String
- qualification : String
- appointedBy : String
- joined : boolean
+ FullTimeStaffHire : constructor
+ getsalary() : int
+ getworkingHour : int
+ getstaffName : String
+ getqualification : String
+ getappointedBy() : String
+ getjoined() : boolean
+ setsalary(int salary) : void
+ setworkingHour(int workingHour) : void
+ hirefulltimestaff() : void
+ display() : void
Part time Staff Hire
- workingHour : int
- wagesperHour : int
- staffName : String
- joiningDate : String
- qualification : String
- appointedBy : String
- shifts : String
- joined : boolean
- terminated : boolean
+ ParttimestaffHire : constructor
+ getworking() : String
+ wagesPerHour() : String
+ getstaffName() : String
+ getjoiningDate() : String
+ getqualification() : String
+ getappouintedBy() : String
+ getshifts() : String
+ getjoined() : boolean
+ getterminated() : boolean
+ setshifts(String shifts) : void
+ hireparttimestaffhire() : void
+ setterminated(String terminated) : void
+ display() : void

Figure 18:Partial class diagram

staff hire

Staff Hire
- JobType : String - Vacancynumber : String - Designation : String
+ getVacancynumber() : String + getDesignation() : String + getJobType() : String + setVacancynumber(String Vaccancynumber) :String + setDesignation(String Designation) : String + setJobType(String JobType) : String + display():void

Figure 19 StaffHire

Full time Staff Hire

Full time Staff Hire
- salary : int - workingHour : int - staffNmae : String - joiningDate : String - qualification : String - appointedBy : String - joined : boolean
+ FullTimeStaffHire : constructor + getsalary() : int + getworkingHour : int + getstaffName : String + getqualification : String + getappointedBy() : String + getjoined() : boolean + setsalary(int salary) : void + setworkingHour(int workingHour) : void + hirefulltimestaff() : void + display() : void

Figure 20 FullTimeStaffHire

Part time Staff Hire

- workingHour : int
- wagesperHour : int
- staffName : String
- joiningDate : String
- qualification : String
- appointedBy : String
- shifts : String
- joined : Boolean
- terminated : boolean

+ ParttimestaffHire : constructor
+ getworking() : String
+ wagesPerHour() : String
+ getstaffName() : String
+ getjoiningDate() : String
+ getqualification() : String
+ getappouintedBy() : String
+ getshifts() : String
+ getjoined() : boolean
+ getterminated() : boolean
+ setshifts(String shifts) : void
+ hireparttimestaffhire() : void
+ setterminated(String terminated) : void
+ display() : void

Figure 21 PartTimeStaffHire

Pseudo code

1.Pseudo code for Staff Hire class:

```
CREATE : StaffHire class

DECLARE class variables :

          Int Vaccancynumber,
          String Designation,
          String Job Type;


FUNCTION getVacancynumber()
     DO

     RETURN Vacancynumber;

END DO



FUNCTION setVacancynumber(int Vacancynumber)

DO

          this.Vacancynumber=Vacancynumber;

END DO


FUNCTION getDesignation()

     DO

          RETURN Designation;

END DO


FUNCTION setDesignation(int Designation)

DO

        this.Designation=Designation;
```

```
END DO

FUNCTION  getJobType()
DO
    RETURN JobType;
END DO

FUNCTION setJobType(String JobType)
DO
        this.JobType=JobType;
END DO

FUNCTION display()
DO
          DISPLAY Vacancy Number
          DISPLAY Designation
          DISPLAY Job Type
END DO
```

2. Pseudo code of Full Time Staff Hire:

```
CREATE : FullTimeStaffHire  EXTENDS StaffHire
```

```
DECLARE class variable:
   int salary;
   int workingHour;
   String staffName;
   String joiningDate;
   String qualification;
   String appointedBy;
   boolean joined;

FUNCTION  FullTimeStaffHire
                SUPER(Vacancynumber,Designation,JobType);
                this.salary=salary;
                this.workingHour=workingHour;
                this.staffName=" ";
                this.joiningDate=" ";
                this.qualification=" ";
                this.appointedBy=" ";

FUNCTION getsalary()
   DO
                RETURN salary;
END DO
```

```
FUNCTION getworkingHour()
        DO
                        RETURN workingHour;
        END DO



FUNCTION  getstaffName()
    DO
                        RETURN staffName;
END DO

FUNCTION getjoiningDate()
                DO
                RETURN joiningDate;
END DO

FUNCTION getqualification()
  DO
                        RETURN qualification;
END DO

FUNCTION getappointedBy()
      DO
```

```
                    **RETURN** appointedBy ;
    **END DO**

**FUNCTION** getjoined()
  **DO**
            **RETURN** joined;
 **END DO**

**FUNCTION** setsalary()
**DO**
    this.salary=salary
    **IF** (joined==true)
    **DISPLAY”** You have been appointed so, cannot change    the salary value” +
    salary
**END IF**
**END DO**
**FUNCTION** setworkingHour(int workingHour)
**DO**
    This.workingHour=workingHour;
    **DISPLAY**” The new working hour is"+workingHour;
**END DO**
**FUNCTION** hirefulltimestaff(String staffName,String joiningDate,String qualification,
String appointedBy)
**DO**
```

```
        this.staffName=staffName;
        this.joiningDate=joiningDate;
IF (joined==true)
        DISPLAY staffName;
        DISPLAY JoiningDate;
        DISPLAY   "This Staff is already appointed";
END IF
ELSE
        this.staffName=staffName;
        this.joiningDate=joiningDate;
        this.qualification=qualification;
        this.appointedBy=appointedBy;
        joined=true;
END ELSE
END DO
FUNCTION  display()
DO
        DISPLAY staffName;
        DISPLAY salay;
        DISPLAY workingHour;
        DISPLAY joiningDate;
        DISPLAY qualification;
        DISPLAY appointedBy;
```

END DO

3. Pseudo code of Part Time Staff Hire:

```
CREATE:PartTimeStaffHire EXTENDS StaffHire
DECLARE  class variable
            int workingHour;
            int wagesPerHour;
            String staffName;
           String joiningDate;
            String qualification;
            String appointedBy;
           String shifts;
            boolean joined;
            boolean terminated;
FUNCTION  parttimestaffhire()
                      SUPER (Vacancynum ariable:ber,Designation,JobType);
                      this.workingHour = workingHour;
                             this.wagesPerHour = wagesPerHour;
                             this.shifts = shifts;
                             this.staffName = "";
                             this.joiningDate = "";
```

```
                    this.qualification = "";

                    this.appointedBy = "";

                    this.joined = false;

                    this.terminated = false;

FUNCTION getworkingHour()

DO

        RETURN workingHour;

END DO

FUNCTION getwagesPerHour()

DO

        RETURN wagesPerHour()

END DO

FUNCTION getstafName()

DO

        RETURN staffName()

END DO

FUNCTION joiningDate()

DO

        RETURN joiningDate()

END DO

FUNCTION getqualification()

DO

        RETURN qualification()
```

```
END DO
FUNCTION getappointedBy()
DO
    RETURN appointedBy()
END DO
FUNCTION getshifts()
DO
    RETURN shifts()
END DO
FUNCTION getterminated()
DO
    RETURN terminated()
END DO
FUNCTION setshifts(String shifts)
DO
    IF (joined==true)
DISPLAY "You have already been apointed so your shifts cannot be changed"
END IF
END DO
FUNCTION hireparttimestaffhire(String staffName,String joiningDate,String qualification,String appointedBy)
  DO
     this.staffName=staffName;
```

```
        this.joiningDate=joiningDate;
        IF (joined==true)

            DISPLAY staffName
            DISPLAY joiningDate
             DISPLAY   "This Staff is already appointed"
        END IF
        ELSE
             this.staffName=staffName;
             this.qualification=qualification;
             this.joiningDate=joiningDate;
             this.appointedBy=appointedBy;
             joined=true;
            terminated=false;
       END ELSE
       END DO
FUNCTION setterminated()
   DO
        IF (this.terminated==true
            DISPLAY "This staff has already been terminated"
       END IF
       ELSE
                staffName="";
                joiningDate="";
                qualification="";
                appointedBy="";
                joined=false;
                terminated=true;
            END ELSE
END DO
FUNCTION display()
DO
              super.display()
        IF(joined==true)
              DISPLAYstaffName
              DISPLAYworkingHour
              DISPLAYjoiningDate
              DISPLAYwagesPerHour
              DISPLAYqualification
              DISPLAY appointedBy
            DISPLAY"Income Per Day="+(wagesPerHour*workingHour);
        END IF
END DO
```

METHOD DESCRIPTION

Method for staff hire

1.Method Name: setVacancynumber(int Vacancynumber)

The word “set” Infront of Vacancynumber (methodname) represents that is a mutator method. The main use of creating this method is that it helps to access the method name of every object of staffHire class. Not only in this method, but also in all the mutator methods this keyword refers to the current object.

2.Method Name: setDesignation(String Designation)

The word “set” in front of the method name represents that this is a mutator method.

Unlike accessor method, this type of method does not return any value and has “void” return type. This method is created in order to change the name of developer whenever needed**.**

3.Method Name: setJobType(String JobType)

So, like as mentioned above it is also an mutator method which contains “set” type of keyword and access name of method of every object. this type of method does not return any value and has “void” return type. This keyword here is used to refer current object

4.Method Name:getVacancynumber()

The word “get” in front of the Vacancynumber(method Name) represents that it is an accessor method. The main use of creating this method is that it helps to access the Vacancynumber of every object of the class StaffHire. Not only in this method, but also in all the accessor methods this keyword refers to the current object.

5.Method Name: getDesignation()

This is an accessor method whose return type is String and returns the value in String, which allows us to access Designation every objects of the class StaffHire.

6.Method Name: getJobType()

The word “get” in front of the JobType (method Name) represents that it is an accessor method. The main use of creating this method is that it helps to access the JobType of every object of the class StaffHire.

7.Method Name: display()

This method is also a mutator method. It has void return type. This method is created in order to print out the information of the objects of this class. Such as Vacancynumber, Designation and JobType if its value is already set.

Method Description for FullTimeStaffHire class

1.Method Name: getsalary()

The word “get” in front of the salary (method Name) represents that it is an accessor method. The main use of creating this method is that it helps to access the salary of every object of the class FulltimeStaffHire. This method has return type as int which access int type value.

2.Method Name: getworkingHour()

The keyword “get” Infront of getworkingHour (method name) represents that it is an accessor method. It contains int types of return type which access int value and returns workingHour.

3.Method Name: getstaffName()

The keyword “get” Infront of staffName (method name) represents that it is an accessor method. It contains String types of return type which access String value and returns It contains String types of return type which access String value and returns staffName.

4.Method Name: getjoiningDate()

The keyword “get” Infront of joiningDate (method name) represents that it is an accessor method. It contains String types of return type which access String value and returns It contains String types of return type which access String value and returns joiningDate.

5.Method Name: getqualification()

The keyword “get” Infront of qualification (method name) represents that it is an accessor method. It contains String types of return type which access String value and returns It contains String types of return type which access String value and returns qualification.

6.Method Name: getappointedBy()

The keyword “get” Infront of appointedBy (method name) represents that it is an accessor method. It contains String types of return type which access String value and returns It contains String types of return type which access String value and returns appointedBy.

7.Method Name: getjoined()

This is an accessor method with Boolean return type and allows us to access whether the person has joined or not and returns a message including true or false.

8.Method Name: setsalary(int salary)

The word “set” in front of the method name represents that this is a mutator method.

Unlike accessor method, this type of method does not return any value and has “void” return type. This method is created to access method name of every object of FullTimeStaffHire. And it has int type of return types.

9.Method Name: setworkingHour(int workingHour)

The word “set” in front of the method name represents that this is a mutator method. The reason for creating this method is to access every object in FullTimeStaffHire. It contains int types of return types of return type.

10.Method Name: hirefulltimestaff(String staffName, String joiningDate,String qualification, String appointedBy)

This method also has “set” keyword which represents so called mutator method. This method was created to analysis the staff has joined or not which has Boolean types of return types. It also check the joining status and print the message as per condition like true or false.

11.Method Name: display()

This method is also a mutator method. It has void return type. This method is created in order to print out the information of the objects of this class. Such as staffName, workingHour, joiningDate, qualification and appointedBy.

Method Description for PartTimeStaffHire class

1.Method Name: getworkingHour()

The word "get" in front of the workingHour (method Name) represents that it is an accessor method. The main use of creating this method is that it helps to access the workingHour of every object of the class. Not only in this method, but also in all the accessor methods this keyword refers to the current object.

2.Method Name:getwagesPerHour()

The word "get" in front of the wagesPerHour (method Name) represents that it is an accessor method. The main use of creating this method is that it helps to access the wagesPerHour of every object of the class. Not only in this method, but also in all the accessor methods this keyword refers to the current object.

3.Method Name: getstaffName()

The word "get" in front of the staffName (method Name) represents that it is an accessor method. The main use of creating this method is that it helps to access the staffName of every object of the class. Not only in this method, but also in all the accessor methods this keyword refers to the current object.

4.Method Name: getjoiningDate()

The word "get" in front of the **joiningDate** (method Name) represents that it is an accessor method. This method has String as return type which indeed returns String value as joiningDate.

5.Method Name: getqualification()

The word "get" in front of the qualification (method Name) represents that it is an accessor method. This method has String as return type which indeed returns String value as qualification.

6.Method Name: getappointedBy()

The word “get” in front of the appointedBy (method Name) represents that it is an accessor method. This method has String as return type which indeed returns String value as appointedBy.

7.Method Name: getshifts()

The word “get” in front of the shifts (method Name) represents that it is an accessor method. This method has String as return type which indeed returns String value as shifts like morning, day or night.

8.Method Name: getjoined()

This is an accessor method with Boolean return type and allows us to access whether the person has joined or not and returns a message including true or false.

9.Method Name: getterminated()

The word “get” in front of the terminated (method Name) represents that it is an accessor method. This method has String as return type which indeed returns String value as terminated. It checks the terminated status of staff and returns a message including true or false.

10.Method Name: setshifts(String shifts)

The word “set” in front of the method name represents that this is a mutator method. Unlike accessor method, this type of method does not return any value and has “void” return type. It contains String as return type and returns which shift like Morning, Day etc.

11.Method Name: hireparttimestaffhire(String staffName, String joiningDate, String qualification, String appointedBy)

This method also has “set” keyword which represents so called mutator method. This method was created to analysis the staff has joined or not which has Boolean types of return types. It also check the joining status and print the message as per condition like true or false.

12.Method Name: void setterminated()

This method also has “set” keyword which represents so called mutator method.it has void types of return types. The reason for creating this method to acess ever object in PartTimeStaffHire and gives output as per condition and termination.

13.Method Name: void display()

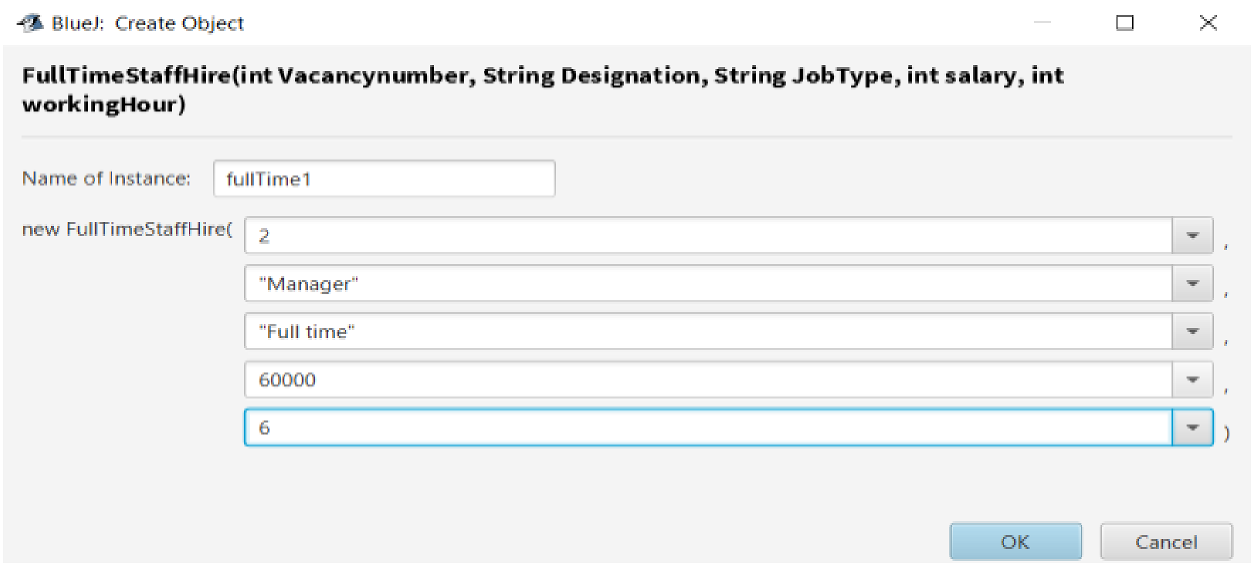
This method is also a mutator method. It has void return type. This method is created in order to print out the information of the objects of this class. Such as staffName, workingHour, joiningDate, Wages Per Hour, qualification, appointedBy and Income Per Day.

TEST

Test 1- To inspect FULLTIMESTAFFHIRE class, appointed the full staff and reinspect the FULLTIME STAFFHIRE CLASS.

Test no: 1	
Objective:	To inspect FullTimeStaffHire and re-inspect the FullTimeStaff class
Action:	➢ FullTimeStaffHire is called with the following arguments: Vacancynumber:2 Designation: “Manager” JobType: “FULL TIME” salary:60000 workingHour:6 ➢ Inspection of the FullTimeStaffHire. ➢ void hirefulltimestaff(String staffName, String joiningDate, String qualification, String appointedBy) satffName:”Muna gurung” joiningDate:”2019/10/03” qualification:”Bachelor” appointedBy:”Anish tamang” ➢ Re-inspection of the FullTimeStaffHire
Expected Result:	The full time staff would be appointed.
Actual Result:	The Full time staff was appointed
Conclusion:	The test is successful.

OUTPUT RESULT:



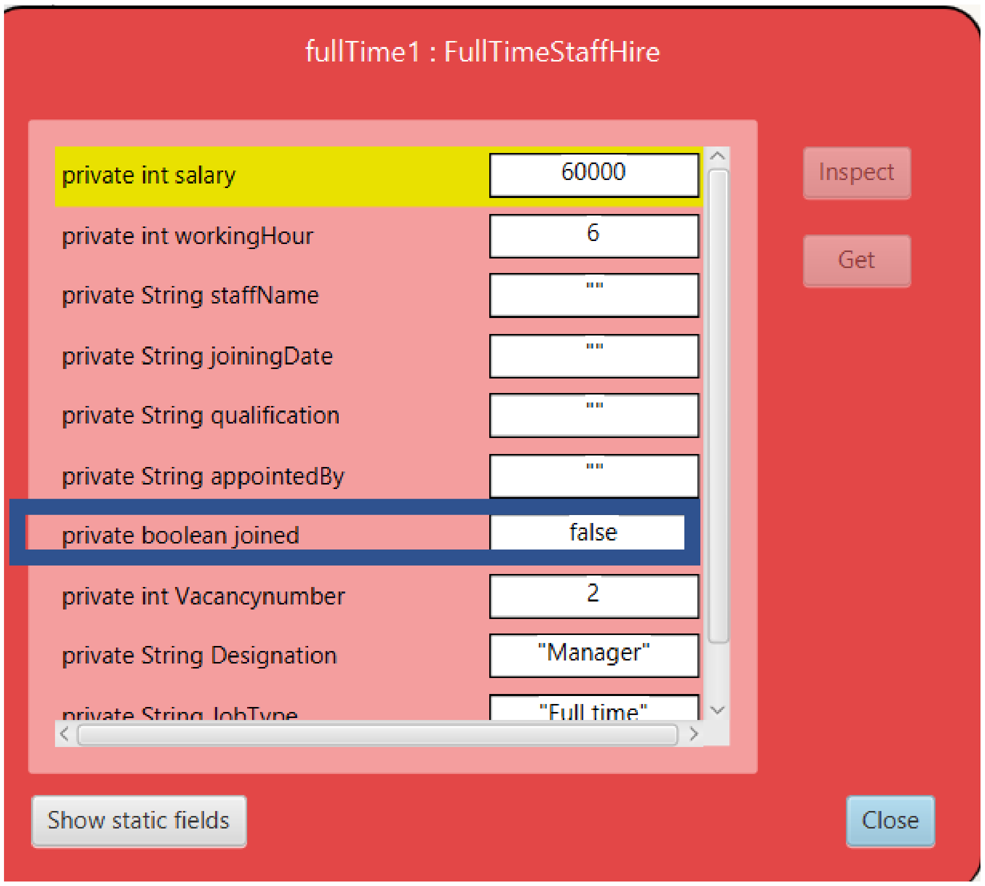
BlueJ: Create Object

FullTimeStaffHire(int Vacancynumber, String Designation, String JobType, int salary, int workingHour)

Name of Instance: fullTime1

new FullTimeStaffHire(2 , "Manager" , "Full time" , 60000 , 6)

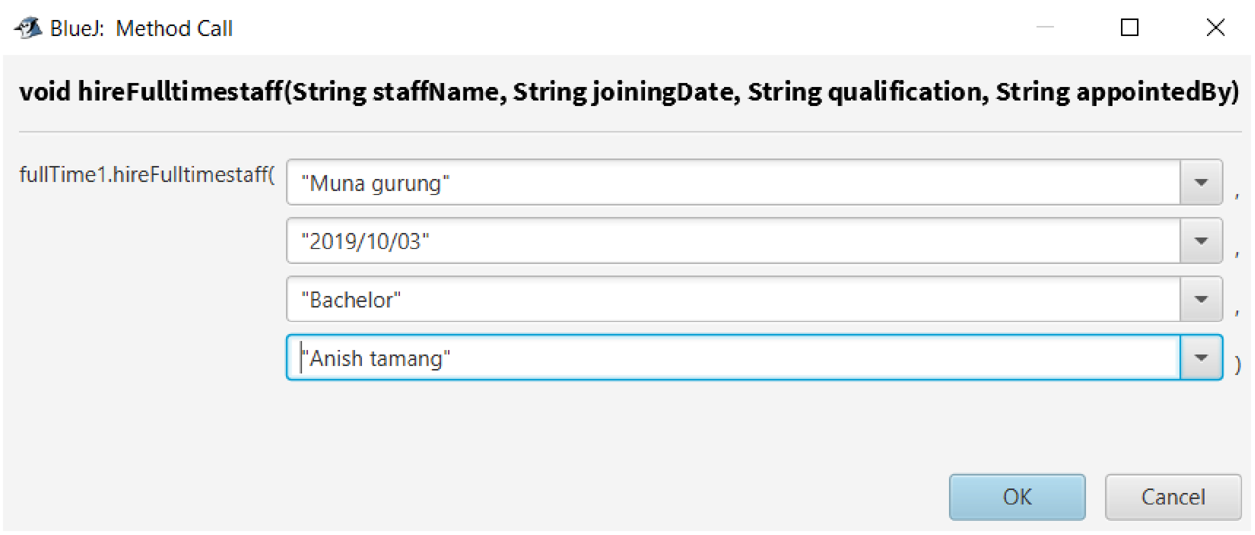
Figure 22 Screenshot of assign the data in fullTimeStaffHire class



fullTime1 : FullTimeStaffHire

private int salary	60000
private int workingHour	6
private String staffName	""
private String joiningDate	""
private String qualification	""
private String appointedBy	""
private boolean joined	false
private int Vacancynumber	2
private String Designation	"Manager"
private String JobType	"Full time"

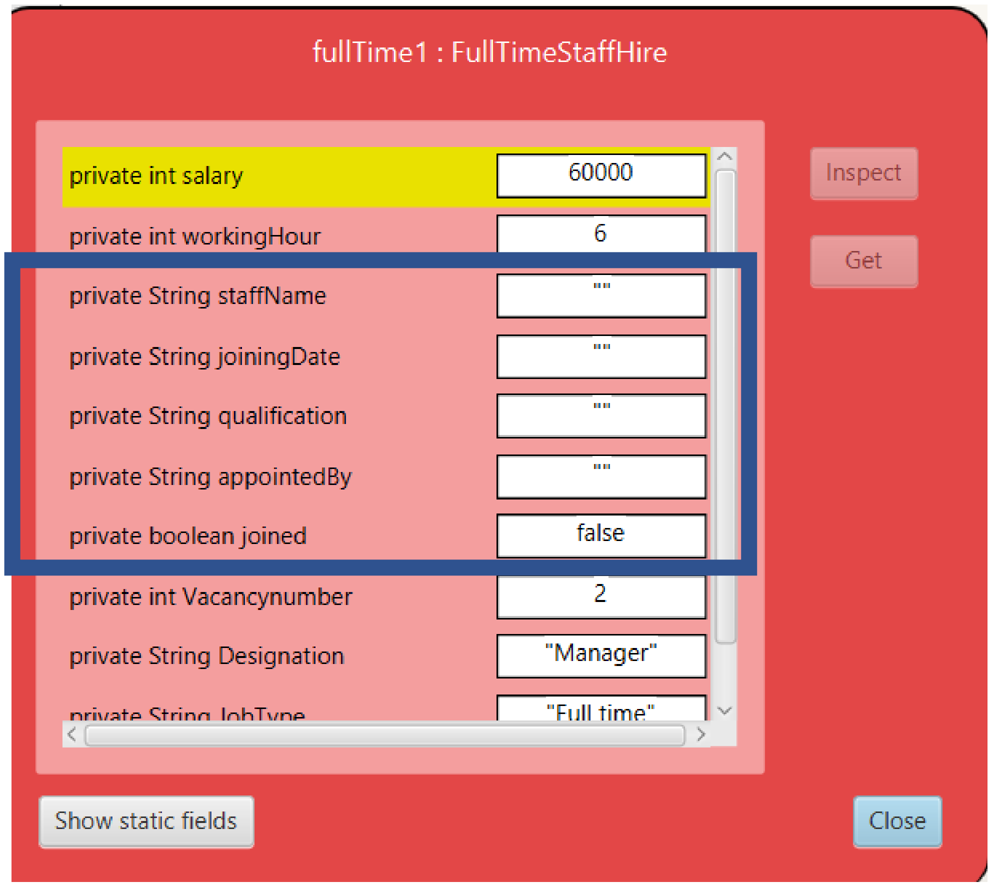
Figure 23 Screenshot for inspecting of Full time staff Hire


BlueJ: Method Call

void hireFulltimestaff(String staffName, String joiningDate, String qualification, String appointedBy)

fullTime1.hireFulltimestaff("Muna gurung" , "2019/10/03" , "Bachelor" , "Anish tamang")

Figure 24Screenshot of assign the data in fullTimeStaffHire class


fullTime1 : FullTimeStaffHire

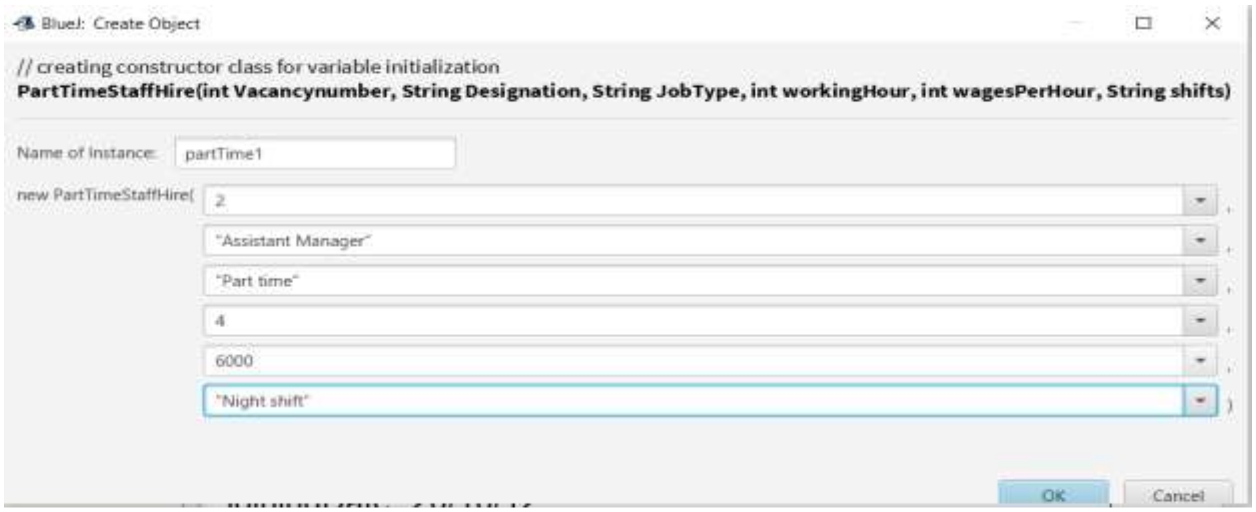
private int salary	60000
private int workingHour	6
private String staffName	""
private String joiningDate	""
private String qualification	""
private String appointedBy	""
private boolean joined	false
private int Vacancynumber	2
private String Designation	"Manager"
private String JobType	"Full time"

Figure 25Screenshot for inspecting of Full time staff Hire

Test 2: Inspect PartTimeStaffHire Class, appoint part time staff, and reinspect the PartTimeStaffHire Class.

TEST NO:	1
Objective:	To PARTTimeStaffHire and re-inspect the PARTTimeStaff class
Action:	➢ The PARTStaffHire is called with the following arguments: Vacancynumber:2 Designation: “ASSISTANT MANAGER” JobType: “PART TIME” wagesPerHour:6000 workingHour:4 shifts: ”Night SHIFT” ➢ Inspection of the FullTimeStaffHire. ➢ void hirefulltimestaff(String staffName, String joiningDate, String qualification, String appointedBy) satffName:”srijan Thapa magar” joiningDate:”20/10/12” qualification:”High school” appointedBy:”Manager” ➢ Re-inspection of the FullTimeStaffHire
Expected Result:	The part time teacher would be appointed.
Actual Result:	The Full time staff was appointed.
Conclusion:	The test is successful.

+



BlueJ: Create Object

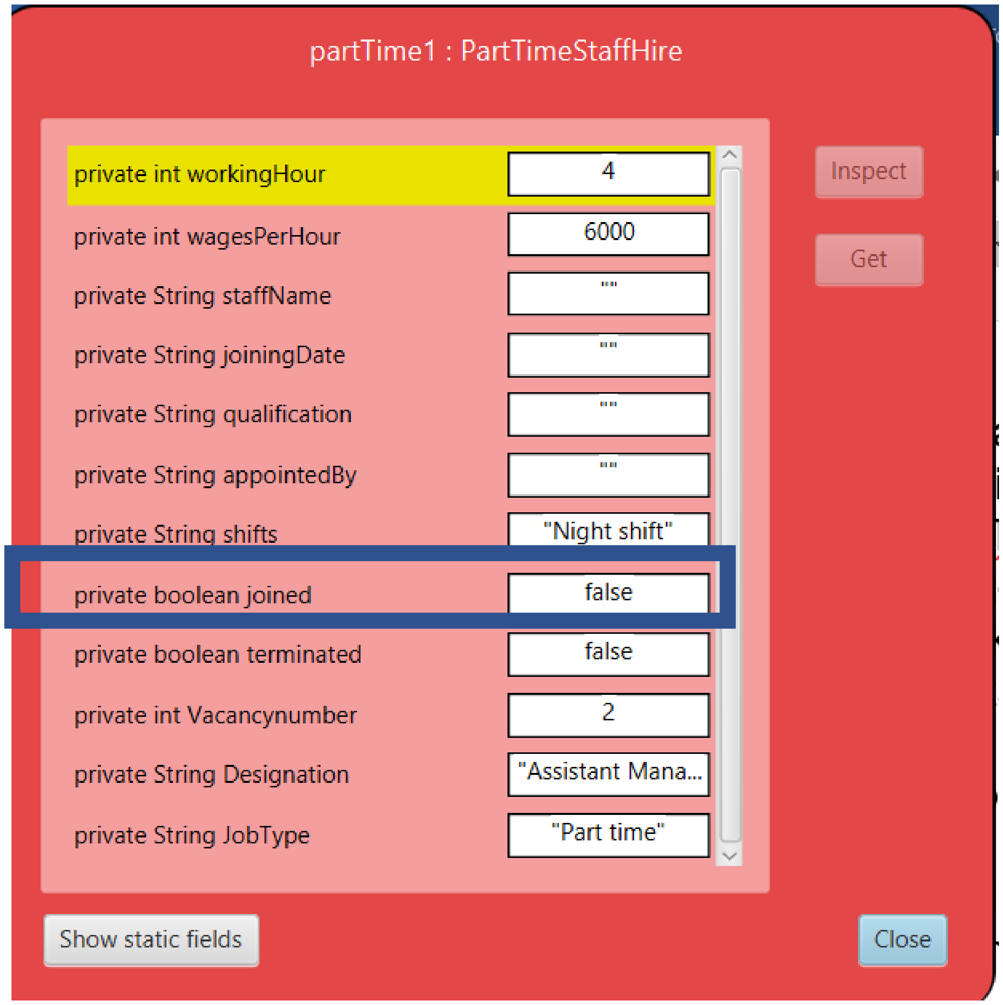
// creating constructor class for variable initialization
PartTimeStaffHire(int Vacancynumber, String Designation, String JobType, int workingHour, int wagesPerHour, String shifts)

Name of Instance: partTime1

new PartTimeStaffHire(2 , "Assistant Manager" , "Part time" , 4 , 6000 , "Night shift")

OK Cancel

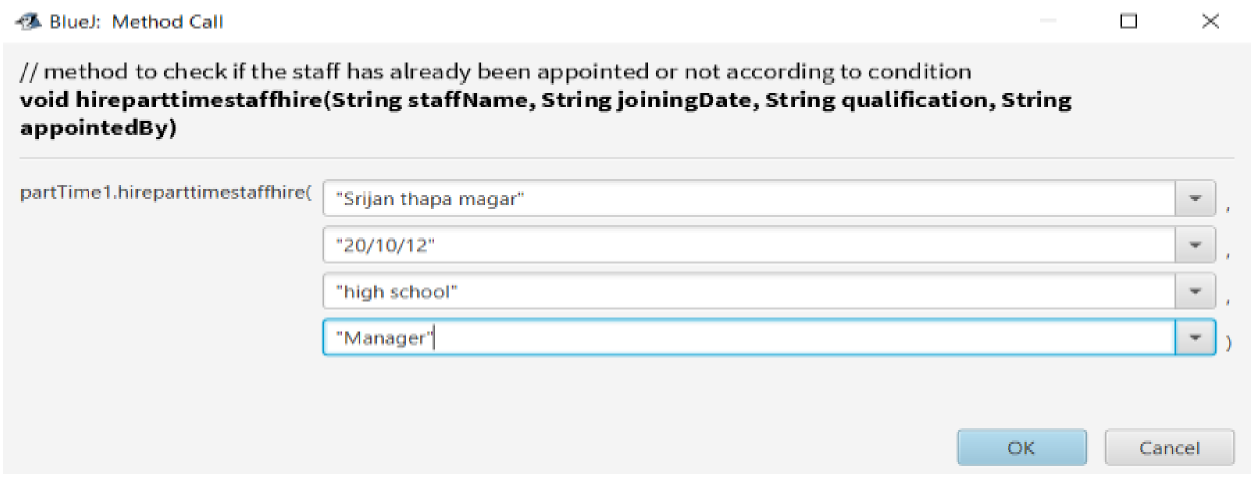
Figure 26 Screenshot for inspecting of Full time staff Hire



partTime1 : PartTimeStaffHire

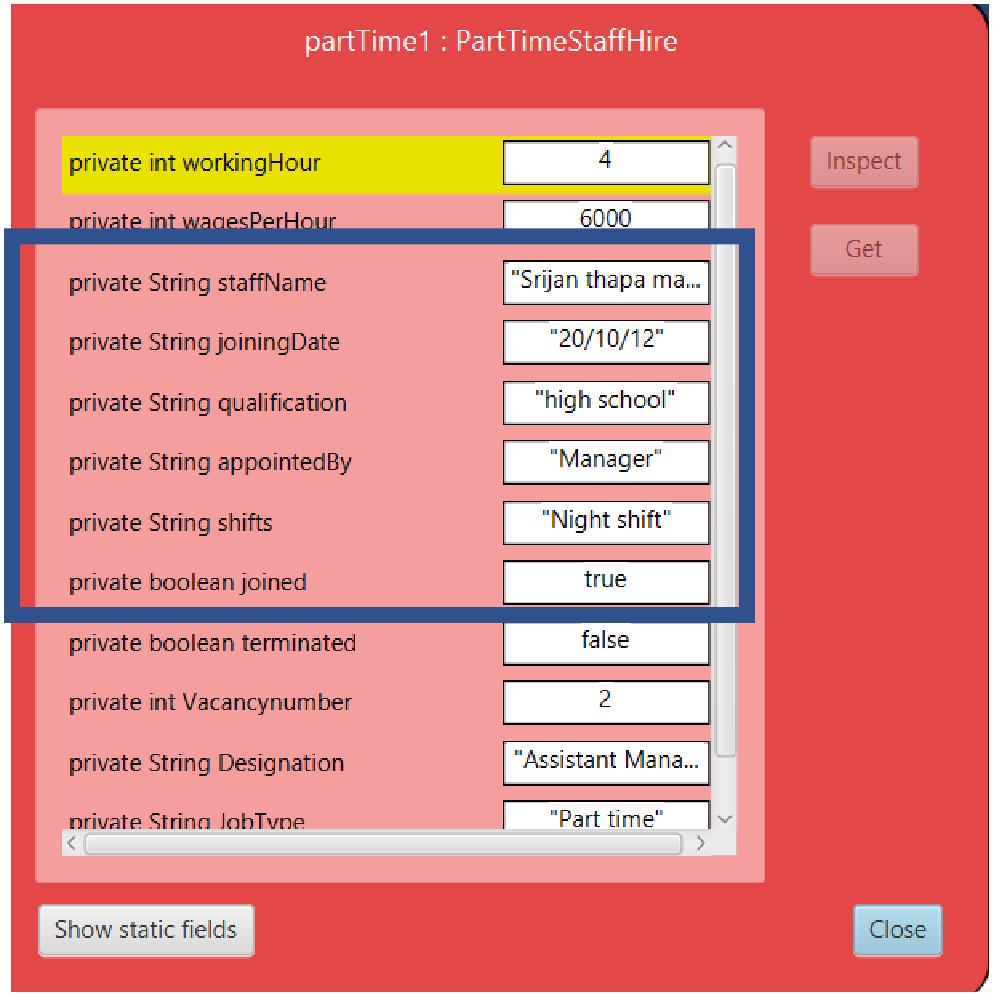
private int workingHour	4
private int wagesPerHour	6000
private String staffName	""
private String joiningDate	""
private String qualification	""
private String appointedBy	""
private String shifts	"Night shift"
private boolean joined	false
private boolean terminated	false
private int Vacancynumber	2
private String Designation	"Assistant Mana...
private String JobType	"Part time"

Figure 27Screenshot for inspecting of part time staff Hire



```
// method to check if the staff has already been appointed or not according to condition
void hireparttimestaffhire(String staffName, String joiningDate, String qualification, String
appointedBy)

partTime1.hireparttimestaffhire( "Srijan thapa magar" ,
                                 "20/10/12" ,
                                 "high school" ,
                                 "Manager" )
```

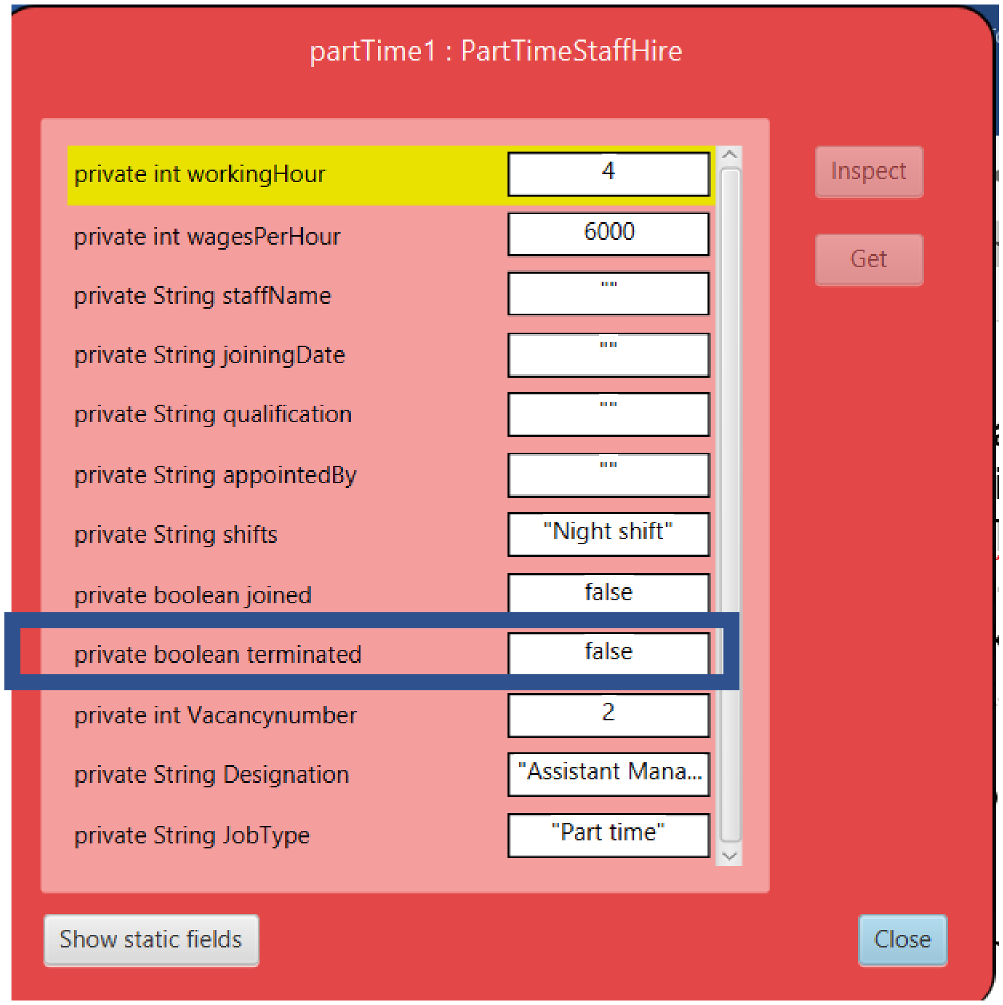
Figure 28 Screenshot for inspecting of Full time staff Hire



```
private int workingHour            4
private int wagesPerHour           6000
private String staffName           "Srijan thapa ma...
private String joiningDate         "20/10/12"
private String qualification       "high school"
private String appointedBy         "Manager"
private String shifts              "Night shift"
private boolean joined             true
private boolean terminated         false
private int Vacancynumber          2
private String Designation         "Assistant Mana...
private String JobType             "Part time"
```

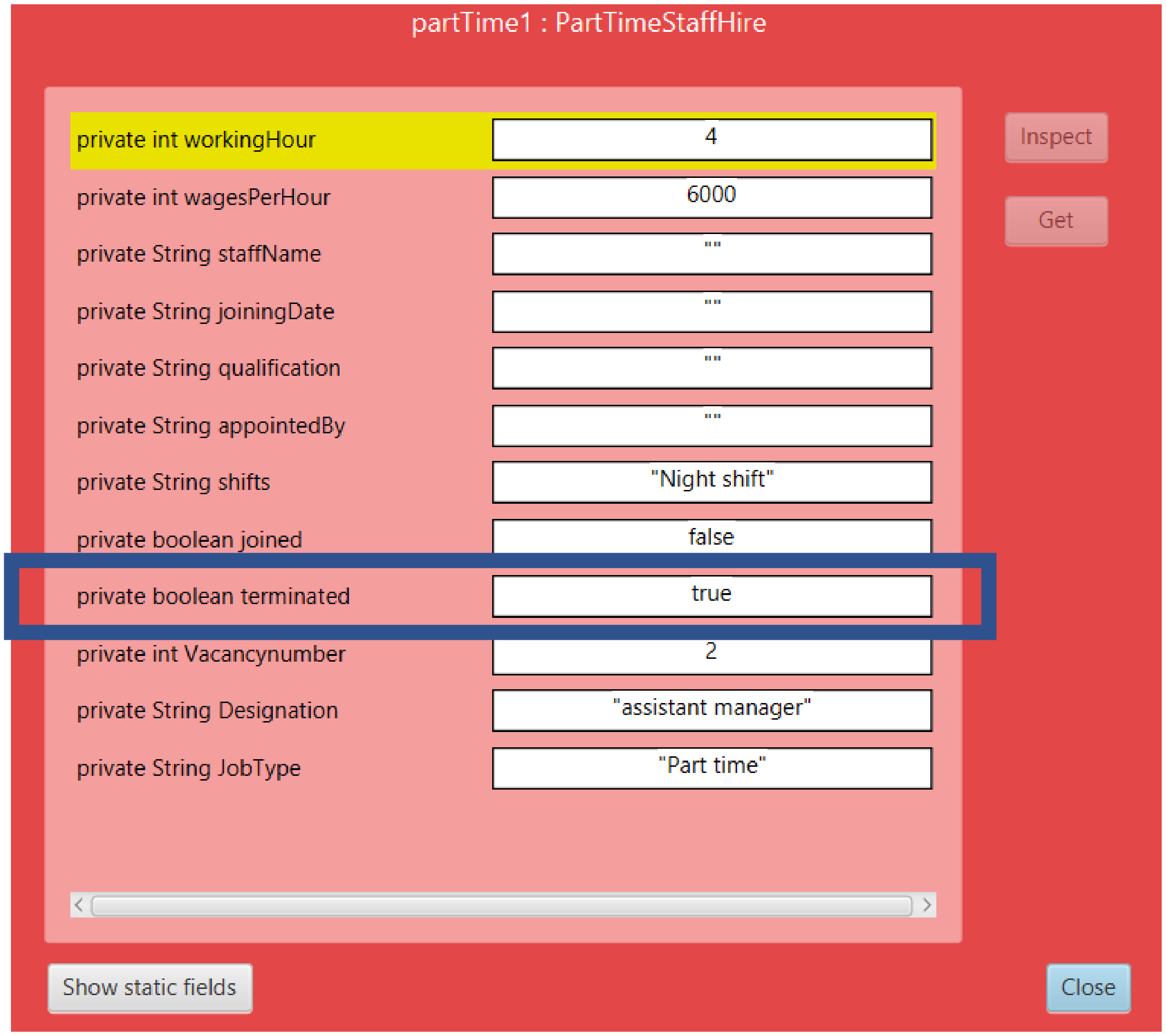
Figure 29 Screenshot for inspecting of part time staff Hire

Test 3: Inspect PartTimeStaffHire Class, change the termination status of a staff, and re-inspect the PartTimeStaffHire Class


partTime1 : PartTimeStaffHire

private int workingHour	4
private int wagesPerHour	6000
private String staffName	""
private String joiningDate	""
private String qualification	""
private String appointedBy	""
private String shifts	"Night shift"
private boolean joined	false
private boolean terminated	false
private int Vacancynumber	2
private String Designation	"Assistant Mana...
private String JobType	"Part time"

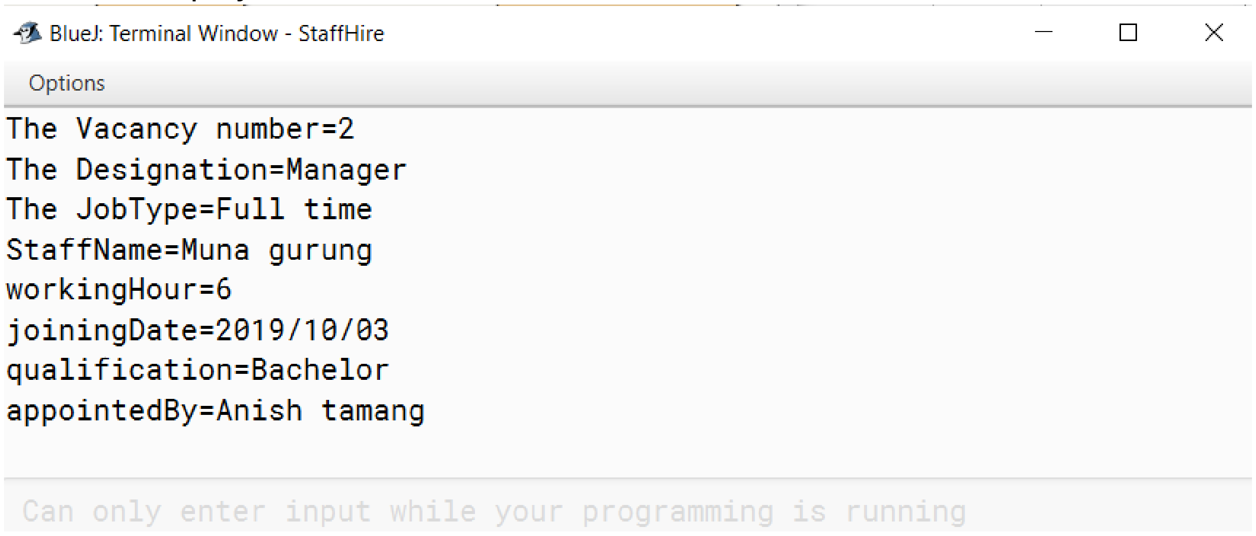
Figure 30 Screenshot of inspecting part time hire



private int workingHour	4
private int wagesPerHour	6000
private String staffName	""
private String joiningDate	""
private String qualification	""
private String appointedBy	""
private String shifts	"Night shift"
private boolean joined	false
private boolean terminated	true
private int Vacancynumber	2
private String Designation	"assistant manager"
private String JobType	"Part time"

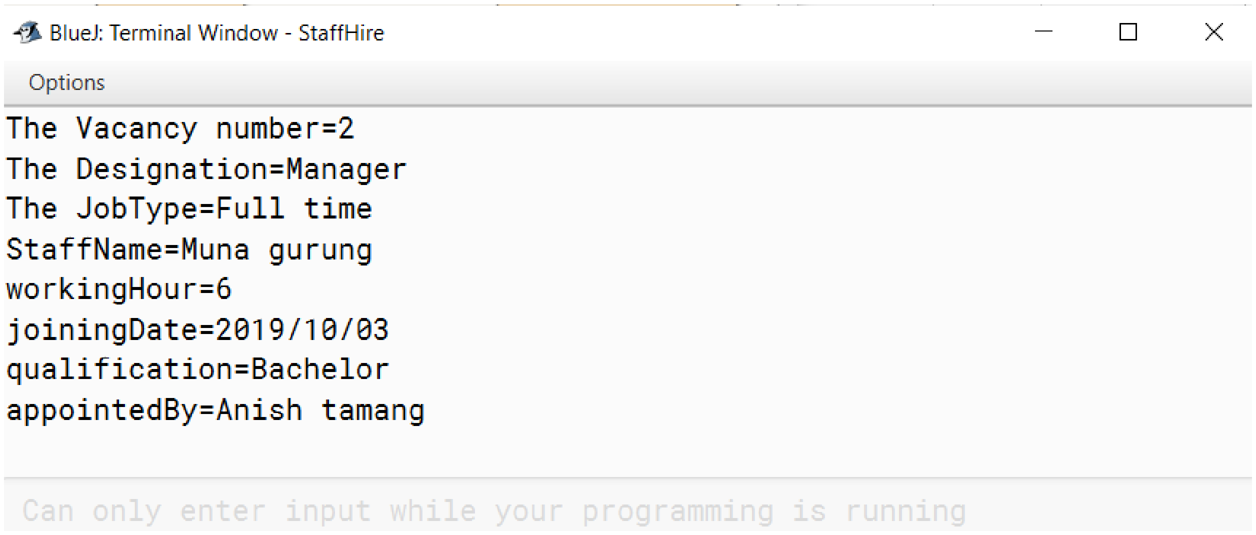
Figure 31 Changing termination status

TEST 4: Display the detail of FullTimeStaffHire and PartTimeStaffHire Class.



```
The Vacancy number=2
The Designation=Manager
The JobType=Full time
StaffName=Muna gurung
workingHour=6
joiningDate=2019/10/03
qualification=Bachelor
appointedBy=Anish tamang
```

Figure 32 Displaying the detail of Fulltime StaffHire

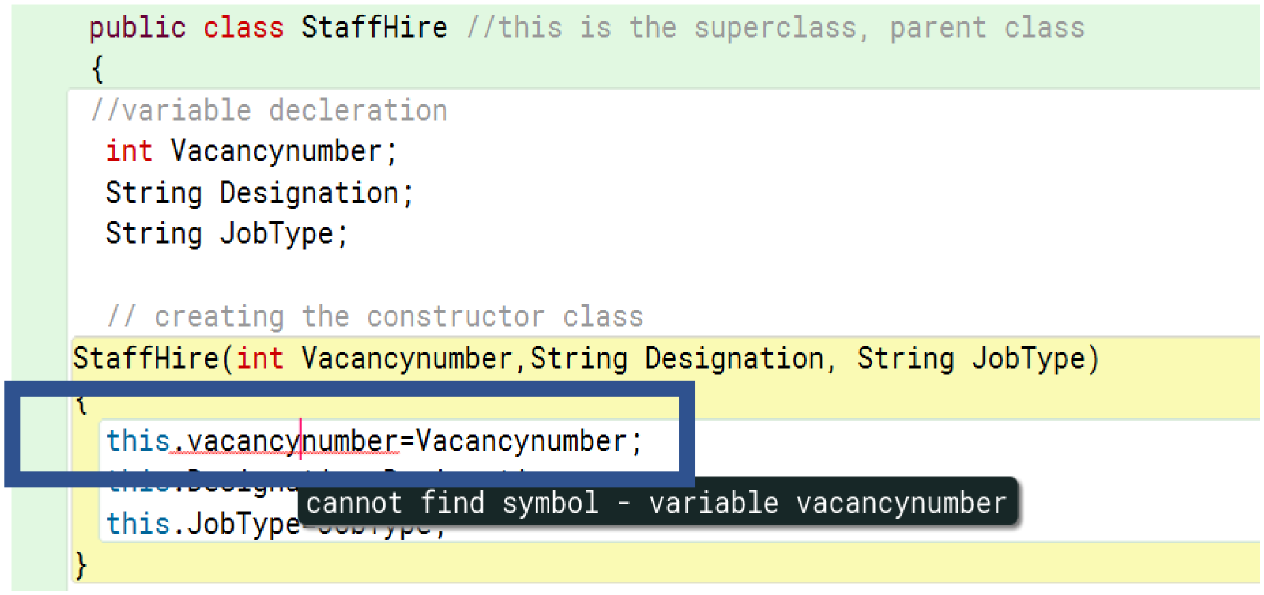


```
The Vacancy number=2
The Designation=Manager
The JobType=Full time
StaffName=Muna gurung
workingHour=6
joiningDate=2019/10/03
qualification=Bachelor
appointedBy=Anish tamang
```

Figure 33 Displaying the detail of parttime staff hire

ERRORS AND CORRECTION

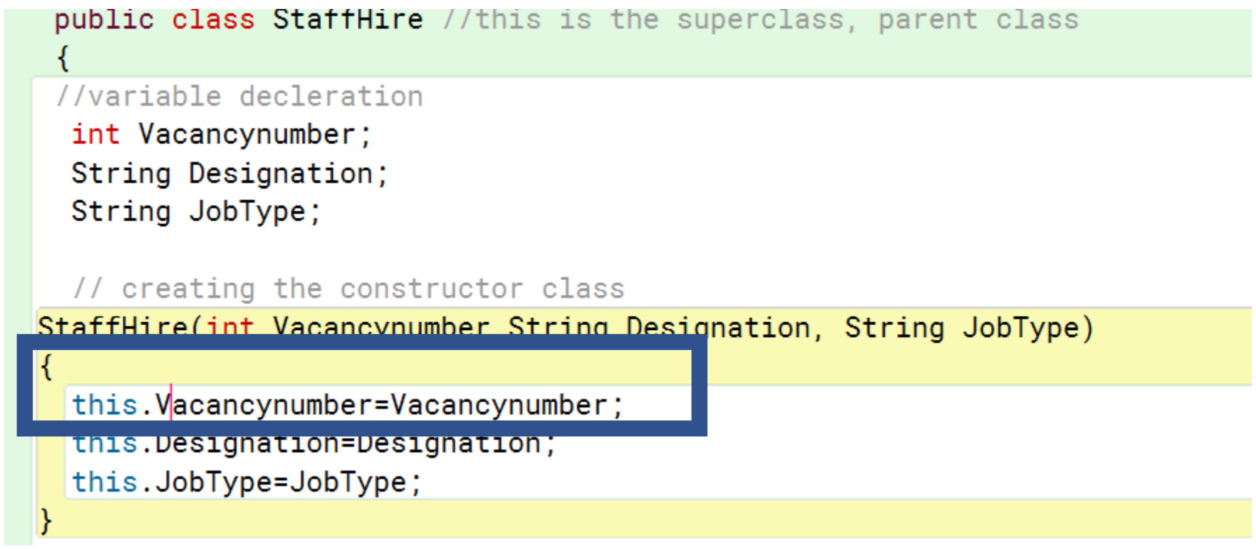
1. Error 1: Syntax error



```
public class StaffHire //this is the superclass, parent class
{
//variable decleration
 int Vacancynumber;
 String Designation;
 String JobType;

 // creating the constructor class
StaffHire(int Vacancynumber,String Designation, String JobType)
{
 this.vacancynumber=Vacancynumber;
 this.Designation=Designation;
 this.JobType=JobType;
}
```

```
cannot find symbol - variable vacancynumber
```

Figure 34 error 1 detection

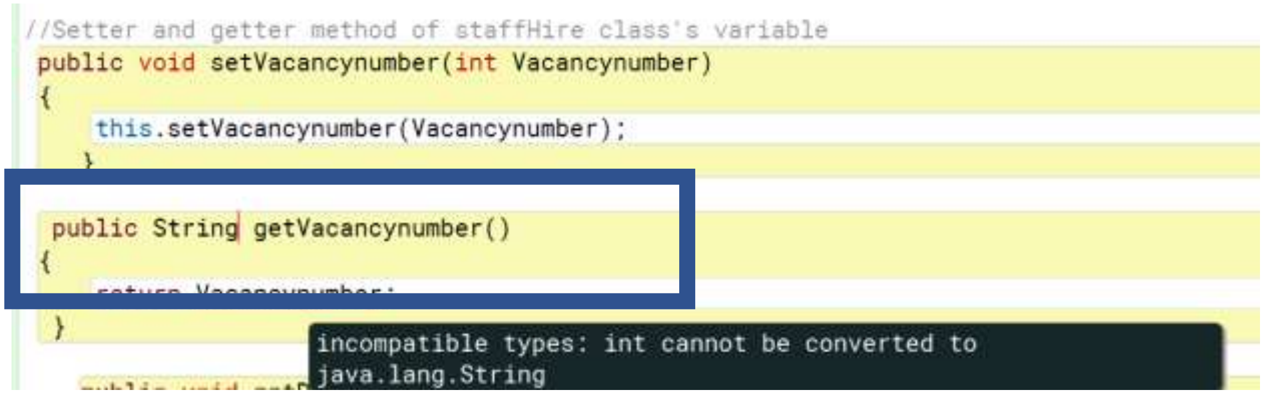


```
public class StaffHire //this is the superclass, parent class
{
//variable decleration
 int Vacancynumber;
 String Designation;
 String JobType;

 // creating the constructor class
StaffHire(int Vacancynumber,String Designation, String JobType)
{
 this.Vacancynumber=Vacancynumber;
 this.Designation=Designation;
 this.JobType=JobType;
}
```

Figure 35 error `1 rectified

While writing code for StaffHire, I did not realized that I had to use String datatype for Vacancynumber. Due to this, I faced problem in further proceeding. While initializing joiningDate, I initialized it as an empty String. However, while using JoiningDate as a parameter for method named hireDeveloper, I use it in the form of integer. Therefore, I faced problem due to datatype incompatibility. Therefore, in order to solve this issue caused due to datatype incompatibility, I went through the codes and changed the datatype of joiningDate from integer to String in method hireDeveloper. Finally, the problem was solved

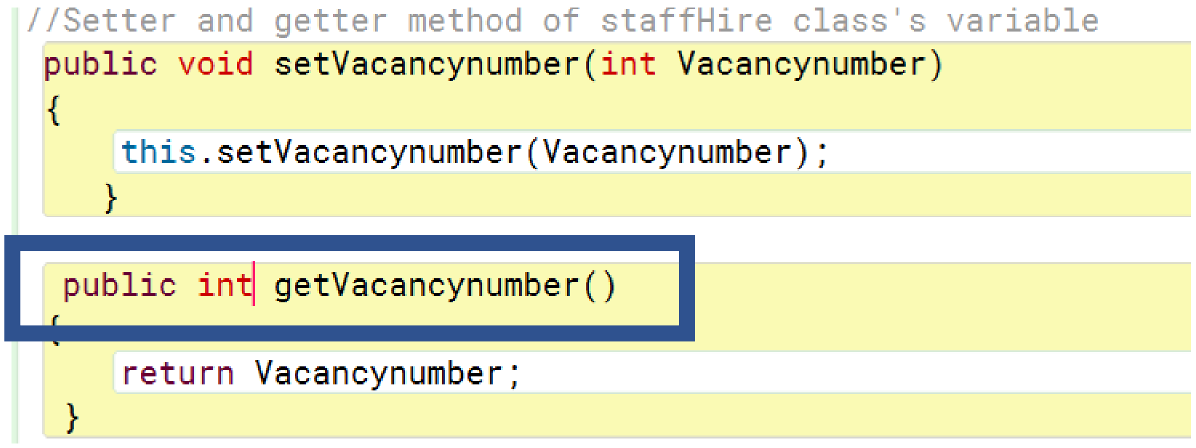
Error 2: Symmetric error



```
//Setter and getter method of staffHire class's variable
public void setVacancynumber(int Vacancynumber)
{
     this.setVacancynumber(Vacancynumber);
    }

 public String getVacancynumber()
{
     return Vacancynumber;
 }
```

```
incompatible types: int cannot be converted to
java.lang.String
```

Figure 36 error 2 detection

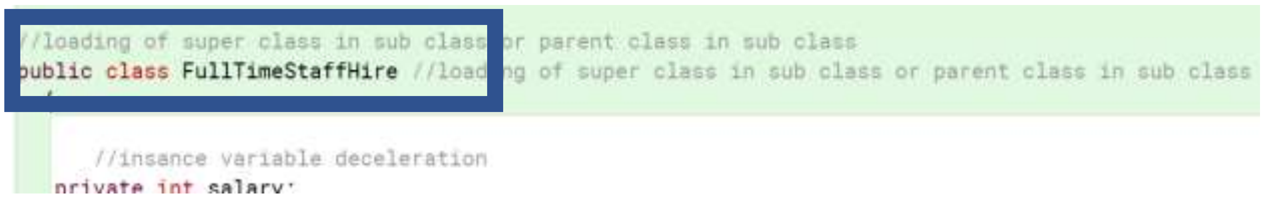


```
//Setter and getter method of staffHire class's variable
public void setVacancynumber(int Vacancynumber)
{
     this.setVacancynumber(Vacancynumber);
    }

 public int getVacancynumber()
{
     return Vacancynumber;
 }
```

Figure 37 error 2 rectified

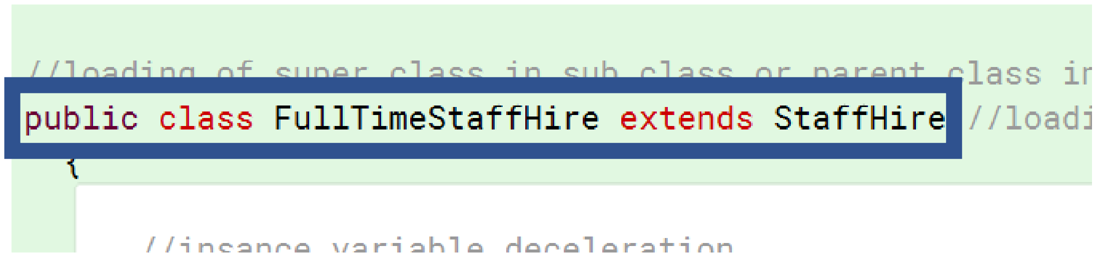
While writing code for StaffHire, I encountered issues while declaring the variable String type of vriable instead of int variable. To overcome this error I made the return same as pervious in variable declaration as above while declaring variables. So, I made the int type of return type.

Extending parent class



```
//loading of super class in sub class or parent class in sub class
public class FullTimeStaffHire //loading of super class in sub class or parent class in sub class
{

    //insance variable deceleration
private int salary;
```

Figure 38 error 3 detection



```
//loading of super class in sub class or parent class in
public class FullTimeStaffHire extends StaffHire //loadi
{

    //insance variable deceleration
```

Figure 39 error 3 rectified

While writing code for Full TImeStaffHire, I encountered an issue while linking this class i.e. FullTImeStaffHire with parent class i.e. StaffHire. It was indicating run-time error in order to solve this problem; I looked at the code that I had written PartTimeStaffHire. Then I realized that I had not extended this class with StaffHire. So, in order to perform correction of this issue, I had a look at the code that I had written for PartTimeStaffHire. Finally, I was able to solve that issue.

Conclusion

As we all know, we cannot compete whole task at a single time and single try. We need to work on it continuously. I read the all scenarios after the release of coursework. It was somehow difficult to understand and confusing too. However, in order to tackle this problem, I performed research on various topics like concept of Inheritance, concept of classes and objects and so on. I gained concrete knowledge on those topics with the help of module teacher, friends and with different websites. Then I started writing code for StaffHire.

Writing a code for StaffHire was not so difficult. But, when I started to write code for FulltimeStaffHire and partTimeStaffHire, I was very frustrated just because scenario was not crystal-clear to me. After that, I took help from my pals and tutor sir in order to handle those problems. Actually, I was stucked in extending FulltimeStaffHire and PartTimeStaffHire. Similarly, I was also stucked in calling methods and instance variables from the parent class. Slowly and steadily, I became able to complete the coding part by grabbing all the helping hands I could get at that time.

After the completion of program, I headed towards checking out of the program that I had developed. I performed different testing. I inspected and re-inspected all the class after creating their particular objects and running a number of methods of their own. I caught quite a number of mistakes in my code and corrected it thus,

To consider my work, I headed closer to trying out the application I have developed. I carried out some of the testing. I inspected all the kind and again I re-inspected all the guidelines after creating positive objects. After re-inspecting I observed some errors in my code and then again accordingly, I corrected them.

This coursework was really interesting as well as challenging too. I learned much more about the concept of OOP (Object Oriented Programming) from this particular coursework. I do believe that we would get much more challenging and interesting coursework in the days to come

APPENDIX

1. STAFF HIRE

```
public class StaffHire   // this is the super class,parent class
{
   //variable decleration
        private int Vacancynumber;
         private String Designation;
        private String JobType;
   // creating the constructor class
   StaffHire(int Vacancynumber,String Designation,String JobType)
   {
     this.Vacancynumber=Vacancynumber;
     this.Designation=Designation;
     this.JobType=JobType;
   }

   //Setter and getter method of staffHire class's variable
   public void setVacancynumber(int Vacacancynumber)
   {
      this.Vacancynumber=Vacacancynumber;
   }
```

```
public int getVacancynumber()
{
   return Vacancynumber;
}

public void setDesignation(String Designation)
{
    this.Designation=Designation;
}

public String getDesignation()
{
   return Designation;
}

public String getJobType()
{
    return JobType;
}

public void setJobType(String JobType)
{
        this.JobType=JobType;
    }
    //end of getter and setter

    public void display()// creating method for value display
    {
        System.out.println("Vacancy Number="+this.getVacancynumber());//callling getters
method for the display
        System.out.println("Designation="+this.getDesignation());
        System.out.println("Job Type="+this.getJobType());
    }
}
```

2. FULL TIME STAFF HIRE:

```
public class FullTimeStaffHire extends StaffHire//loading of super class in sub class or
parent class in sub class
{
    //insance variable deceleration
    private int salary;
    private int workingHour;
    private String staffName;
    private String joiningDate;
    private String qualification;
    private String appointedBy;
    private boolean joined;

    //creating constructor class
    FullTimeStaffHire( int Vacancynumber, String Designation,String JobType,int salary,int
workingHour)
    {
        super(Vacancynumber,Designation,JobType);//calling super class variable
        this.salary=salary;
        this.workingHour=workingHour;
        this.staffName="";
        this.joiningDate="";
        this.qualification="";
        this.appointedBy="";
    }

    //getter method
    public int getsalary()
    {
         return salary;
    }

    public int getworkingHour()
```

```
{
    return workingHour;
}

public String getstaffName()
{
    return staffName;
}

public String getjoiningDate()
{
    return joiningDate;
}

public String getqualification()
{
    return qualification;
}

public String  getappointedBy()
{
    return appointedBy ;
}
```

```
public boolean getjoined()
{
      return joined;
}

/**setter method for values changing according to condition*/
 public void  setsalary(int salary)
 {
  this.salary=salary;
  if (joined==true)
  {
    System.out.println("You have been appointed so, cannot change the salary value"+salary);
  }
}

/**setter method for values changing according to condition*/
public void setworkingHour(int workingHour)
{
   this.workingHour=workingHour;
   System.out.println("The new working hour is"+workingHour);
}


/**new method for value changing according to conditions*/
public void hirefulltimestaff(String staffName,String joiningDate,String qualification, String appointedBy)
{
   this.staffName=staffName;
   this.joiningDate=joiningDate;
  if(joined==true)
  {
      System.out.println("Staff Name="+staffName);
      System.out.println("Joining Date="+joiningDate);
      System.out.println("This Staff is already appointed");
  }
  else
  {
      this.staffName=staffName;
      this.joiningDate=joiningDate;
      this.qualification=qualification;
      this.appointedBy=appointedBy;
      joined=true;


  }
```

```
    }


    //display method for displaying attributes values
    public void display()
    {
        super.display();
        if(joined==true)
        {
            System.out.println("Staff Name="+this.getstaffName());//calling of getters method
            System.out.println("Salary="+this.getsalary());
            System.out.println("Working Hour="+this.getworkingHour());
            System.out.println("Joining Date="+this.getjoiningDate());
            System.out.println("Qualification="+this.getqualification());
            System.out.println("Appointed by="+this.getappointedBy());
        }
    }
}
```

3. PART TIME STAFF HIRE

```
public class PartTimeStaffHire extends StaffHire//loading of super class in sub class or parent class in sub class
{
```

```
//variable decleration
    private int workingHour;
    private int wagesPerHour;
    private String staffName;
    private String joiningDate;
    private String qualification;
    private String appointedBy;
    private String shifts;
    private boolean joined;
    private boolean terminated;

/**
*creating constructor class for variable initialization
*/
PartTimeStaffHire(int Vacancynumber,String Designation,String JobType,int
workingHour,int wagesPerHour,String shifts)
{
   super(Vacancynumber,Designation,JobType);//accessing super class variables
   this.workingHour = workingHour;
   this.wagesPerHour = wagesPerHour;
   this.shifts = shifts;
   this.staffName = "";
   this.joiningDate = "";
```

```
        this.qualification = "";

        this.appointedBy = "";

        this.joined = false;

        this.terminated = false;

    }


        //getter method deceleration

    public int getworkingHour()

        {

            return workingHour;

        }


        public int getwagesPerHour()

        {

            return wagesPerHour;

        }


        public String getstaffName()

        {

             return staffName;

        }


        public String getjoiningDate()
```

```
{
    return joiningDate ;
}

public String getqualification()
{
    return qualification;
}

public String  getappointedBy()
{
   return appointedBy;
}

public String getshifts()
{
   return shifts;
}

public boolean getjoined()
{
   return joined;
}
```

```
public boolean getterminated()
{
   return terminated;
}

/**
 * setter method for value changing according to given condition
 */
public void setshifts(String shifts)
{
    this.shifts=shifts;
    if(joined==true)
    {
        System.out.println("You have already been apointed so your shifts cannot be changed");
    }
}

/**
 * method to check if the staff has already been appointed or not according to condition
 */
```

```
    public void hireparttimestaffhire(String staffName,String joiningDate,String qualification,String appointedBy)
    {
       this.staffName=staffName;
       this.joiningDate=joiningDate;
       if(joined==true)
       {
          System.out.println("Staff Name="+this.staffName);
          System.out.println("Joining Date="+this.joiningDate);
          System.out.println("This Staff is already appointed");
       }
       else
       {
          this.staffName=staffName;
          this.qualification=qualification;
          this.joiningDate=joiningDate;
          this.appointedBy=appointedBy;
          joined=true;
          terminated=false;
       }
    }

    /**
```

```
 * method for staff has been appointed and print suitable output of the condition
 * else takes the new values
 */
public void setterminated()
{
    if(this.terminated==true)
    {
        System.out.println("This staff has already been terminated");
    }
    else
    {
        staffName="";
        joiningDate="";
        qualification="";
        appointedBy="";
        joined=false;
        terminated=true;
    }
}

/**
 * display method to print all output of given condition
 */
```

```
    public void display()
    {
        super.display();//calling super class display method
        if(joined==true)
        {
            System.out.println("Staff Name="+this.getstaffName());//calling getter method
for the dispaly
            System.out.println("Working Hour="+this.getworkingHour());
            System.out.println("Joining Date="+this.getjoiningDate());
            System.out.println("Wages Per Hour="+getwagesPerHour());
            System.out.println("Qualification="+this. getqualification());
            System.out.println("Appointed by="+this.getappointedBy());
            System.out.println("Income Per Day="+(wagesPerHour*workingHour));
        }
    }
```