LONDON METROPOLITAN UNIVERSITY

islington college
(इस्लिङ्टन कलेज)

CS4001NI - PROGRAMMING

30% Individual Coursework

2019-20 Autumn

Student Name: Himanshu Pandey

London Met ID: 19031311

College ID: NP01NT4A190131

Assignment Due Date: 5th June2020

Assignment Submission Date: 5th June2020

I confirm that I understand my coursework needs to be submitted online via Google Classroom under the relevant module page before the deadline in order for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a marks of zero will be awarded.

Contents

FIGURES

1. Introduction

GUI programming involves the use of a number of predefined components such as buttons, checkboxes, text fields, windows, menus, etc that are part of the class hierarchy. A GUI is a collection of instances of these components. (Anon., n.d.)

In Java, there are two sets of GUI components. Prior to Java version 1.2, Java GUIs were build with components from the Abstract Windows Toolkit (AWT). The AWT components are contained within the java.awt package. Java applications using AWT display differently on each platform.

With Java 1.2, a more flexible Swing API was introduced. Swing GUI components allow the designer to specify a uniform look and feel for a GUI application across all platforms, or to use each platforms custom look and feel. Although we will focus on Swing components, Swing and AWT are closely related, because Swing relies on features of the AWT. The principles for using both components are similar.

Using a Graphical User Interface, when you perform an action, an event is generated. In event-driven programming, the program responds to the events.

The GUI programs that we will create in Java will contain three types of software. These are:

1. **Graphical components** that make up the GUI
2. **Listener classes/methods** that receive the events and respond to them, and
3. **Application methods** that do the work for the user.

The graphical components are those like the ones listed above. You can use them as they are, or extend them into your own classes. Listener methods are the methods in your application that respond to different events. Listener methods invoke application methods. An application method is a standard Java method to perform useful tasks. In Java applications, generally these three types of software should be kept separate (reduce coupling).

The class **Component** implements some useful public methods, which are inherited by its subclasses, such as:

- public void setSize(int width, int height) – used to set the size of a component
- public void setBackground(Color c) - used to set the background color of a component
- public void setVisible(boolean b) - used to set the visibility of a component (i.e. to make a component appear/display component on the screen)

The class **Container** is a subclass of **Component**. Components are attached to Containers (such as windows) in order for components to be organized and displayed on the screen. Any object that is a Container can be used to organize other components in a GUI. Because a Container is-a Component, you can attach Containers to other Containers. In other words, a Container is a component that can hold other Containers.

There are two types of **Containers**, panels and windows. Windows are the top level element displayed by an operating system. Panels are containers that cluster related components within a larger object, such as an applet, or application window.

The Container class is an ***abstract class*** which implements some useful methods, such as:

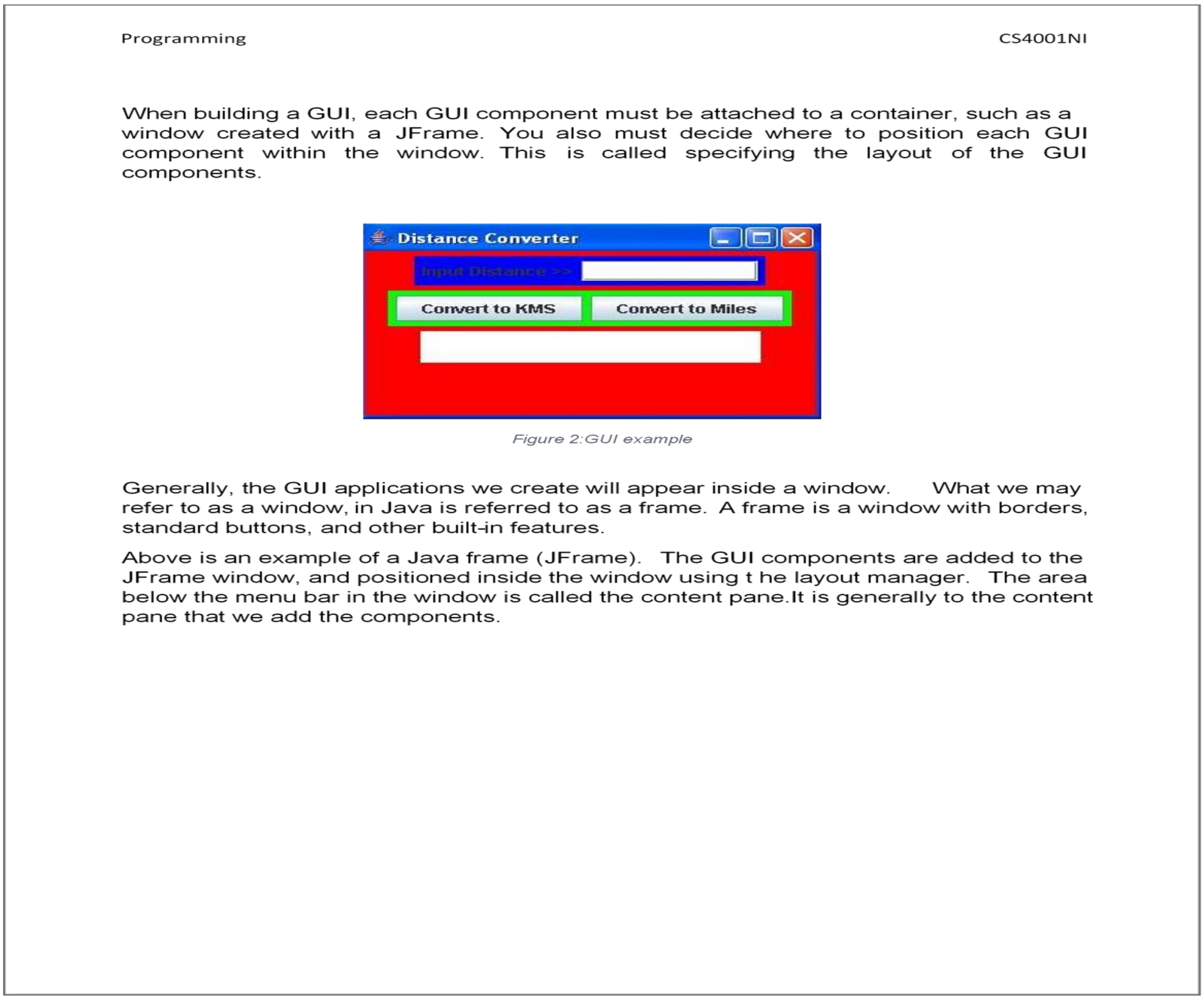
- public Component add(Component) – used to add components to a container
- public setLayout(LayoutManager mgr) – used to set the Layout of a container.

The class JComponent is a subclass of Container. JComponent is the superclass of all lightweight Swing components and declares their common attributes and behaviours. Because a JComponent is a Container, all Swing components are also Containers.

2.Creating a GUI

Generally a GUI **component** is an object that represents a screen element that is used to display information or to allow the user to interact with the program in a certain way. Java components include labels, buttons, text fields, scroll bars, menus etc.

A **container** is a special type of component that is used to hold and organize other components. Frames and Panels are examples of Java containers.



Programming CS4001NI

When building a GUI, each GUI component must be attached to a container, such as a window created with a JFrame. You also must decide where to position each GUI component within the window. This is called specifying the layout of the GUI components.

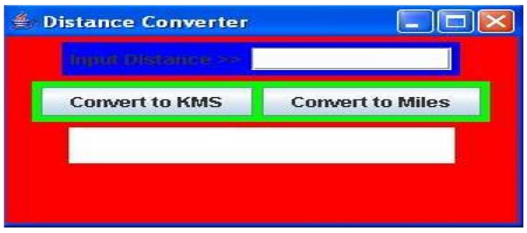


Figure 2:GUI example

Generally, the GUI applications we create will appear inside a window. What we may refer to as a window, in Java is referred to as a frame. A frame is a window with borders, standard buttons, and other built-in features.

Above is an example of a Java frame (JFrame). The GUI components are added to the JFrame window, and positioned inside the window using t he layout manager. The area below the menu bar in the window is called the content pane.It is generally to the content pane that we add the components.

3. Class Diagram

ING NEPAL
-+JFrame myframe; +ArrayList<StaffHire>list=newArrayList<StaffHire>(); +part time staff hire +full time staff hire; +Appoint fulltime staff hire; -JTextField vacancy number; -JTextFeild staffname; -JTextFeild salary; -JTextFeild working hour; -JTextFeild designation; -JTextFeild wages per hour; -JButton appointed by; -JButtonappoint full time staffhire; -JButton appointed partime staffhire; -JButton display; -JButton terminate; -JButton clear; +my Form.void;

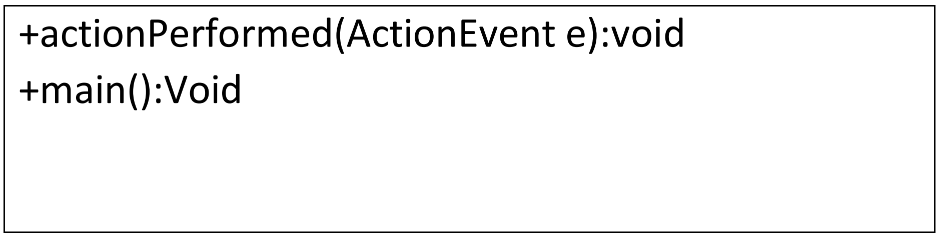


Figure 1 class diagram of ING nepal

Pseudocode:

```
  /**
         * Write a description of class INGNepal here.
         *
         * @author (your name)Himanshu pandey
         * @version ( fiday 5th june 2020)
         */
        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,lblVacancyNumber1,
lblDesignaion1,lblJobType1, lblSalary1,

lblWorkingHour1,lblVacancyNumberApnt1,lblshifts,lblStaffName1,lblJoinDate1,lblQualificati
on1,lblApointedBy1;

          JTextField          txtVacancyNumber,          txtDesignation,txtJobType,
txtSalary,txtWorkingHour,txtVacancyNumberApnt,
             txtStaffName,txtJoinDate,txtQualification,txtApointedBy,txtVacancyNumber1,
txtDesignation1,txtJobTyp1e, txtSalary1,
        txtWorkingHour1,txtVacancyNumberApnt1,
        txtStaffName1,txtJoinDate1,txtQualification1,txtApointedBy1,txtshifts;

    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(221,30,130,30);
        frm.add(lblDesignation);

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

        lblJobType=new JLabel("JobType:");
        lblJobType.setBounds(412,30,80,30);
        frm.add(lblJobType);

        String jobType[]={"FullTime","PartTime"};
        cmbJobType=new JComboBox(jobType);
```

```
cmbJobType.setBounds(479,30,80,30);
frm.add(cmbJobType);

lblSalary=new JLabel("Salary:");
lblSalary.setBounds(20,65,121,30);
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(304,65,100,30);
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()==btnAppointFT1)
     {
         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,appointedBy1);
                           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();
       }
   }
```

4. Short Descriptions of Methods

1. **myForm()**

   This method provides the layout of the GUI that We have made. A GUI program consists of a collection of graphical components that are placed inside one or more windows/applets/panes. Most components are "contained" within a window. Therefore, the window acts as a container to hold various GUI components. A container is an area on the screen that contains smaller areas. I.e. the window is a container, which contains components such as buttons, menus, scroll bars etc.

2. **actionPerformed(ActionEvent e)**

   A button listener must implement the ActionListener interface. ActionListener is an interface (not a class) that contains a single method:

public void actionPerformed(ActionEvent evt) ;

   A class that *implements* the interface must contain an actionPerformed() method. The ActionEvent parameter is an Event object that represents an event (a button click). It contains information about the event.

   Import the package java.awt.event.* if you are dealing with events.

The listener object could be defined in a class other than the frame class. In our example, the class that holds the component is also the listener for its events.

3.**Main()**

This method is to display the input of GUI that is the developers appointed, their salary, specialization, platform, appointed by, staff name, appointed date, termination date and salary.

4. Testing

Test1

Objective	Adding details to full time staff
Action	Added details
Expected Result	Details to be added
Actual Result	Details were added
Conclusion	Test successful

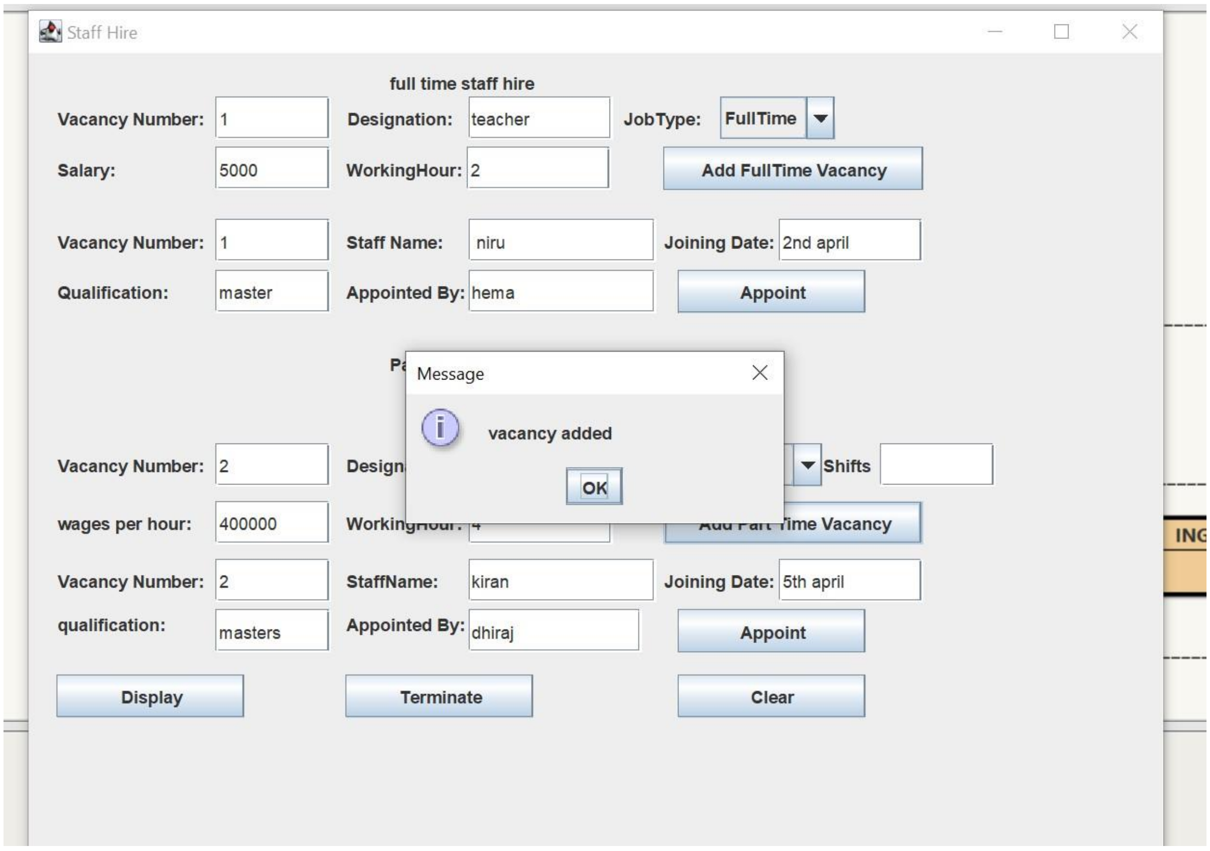
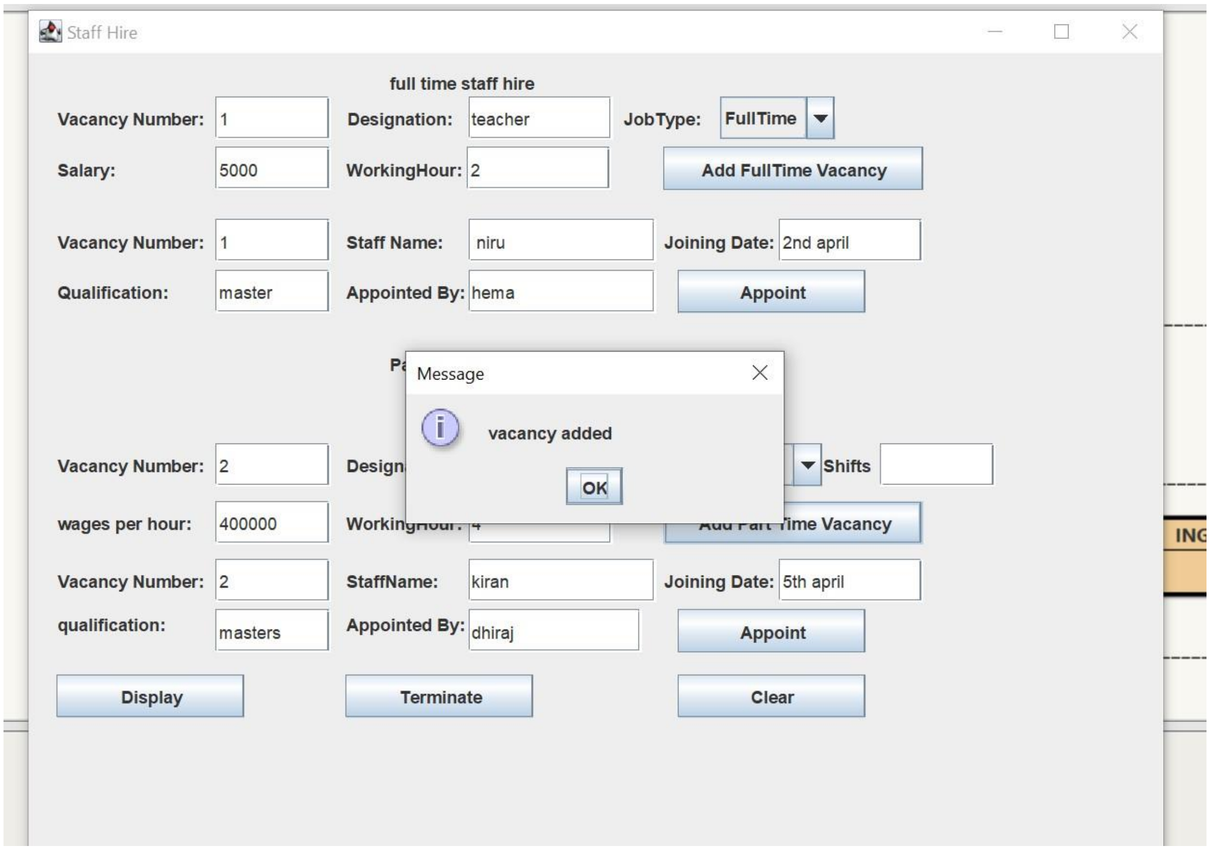


Figure 2 test result of Full time Staff

Test 2

Objective	Adding details to part time staff
Action	Added details
Expected Result	Details to be added
Actual Result	Details were added
Conclusion	Test successful


Staff Hire

full time staff hire

Vacancy Number: 1
Designation: teacher
JobType: FullTime
Salary: 5000
WorkingHour: 2
Add FullTime Vacancy

Vacancy Number: 1
Staff Name: niru
Joining Date: 2nd april
Qualification: master
Appointed By: hema
Appoint

Message
vacancy added
OK

Vacancy Number: 2
Shifts
wages per hour: 400000
Add Part Time Vacancy

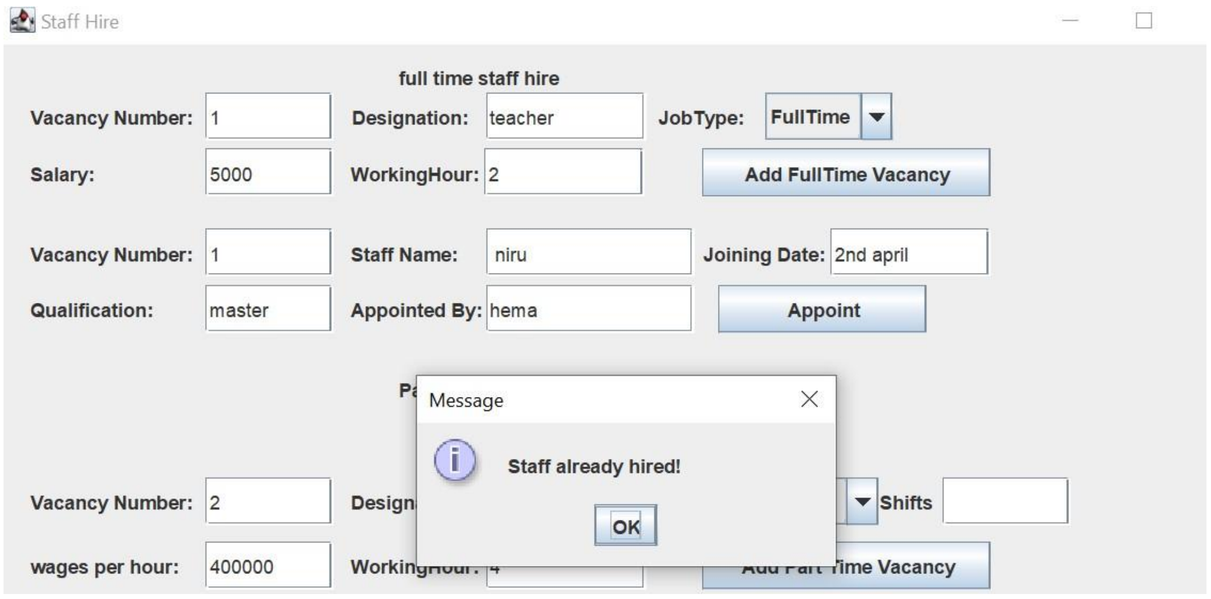
Vacancy Number: 2
StaffName: kiran
Joining Date: 5th april
qualification: masters
Appointed By: dhiraj
Appoint

Display
Terminate
Clear

Figure 3test result of Part time staff

Test 3

Objective	Adding details to appoint full time staff
Action	Added details
Expected Result	Details to be added
Actual Result	Details were added
Conclusion	Test successful


Staff Hire

full time staff hire

Vacancy Number: 1 Designation: teacher JobType: FullTime

Salary: 5000 WorkingHour: 2

Vacancy Number: 1 Staff Name: niru Joining Date: 2nd april

Qualification: master Appointed By: hema

Message

Staff already hired!

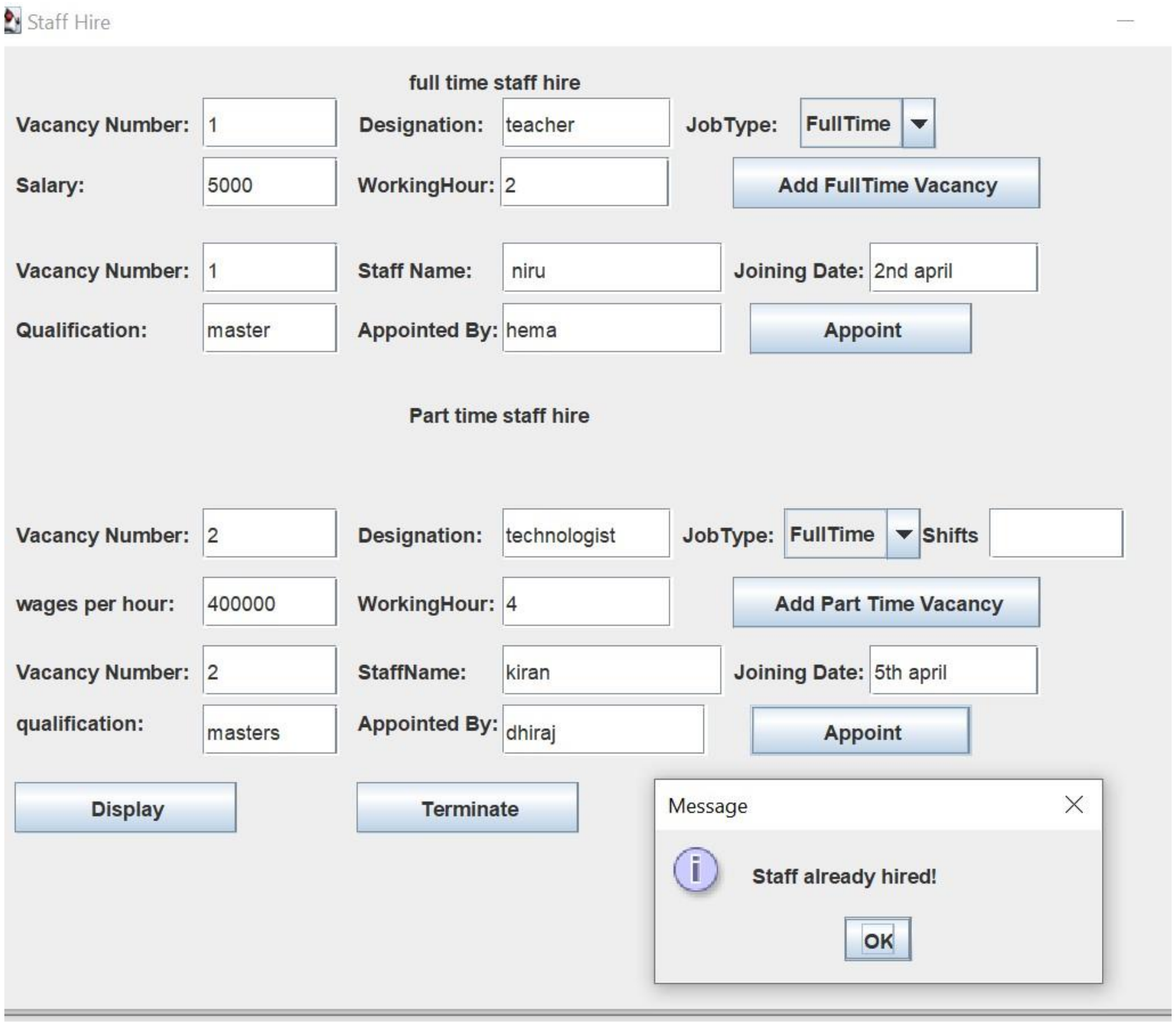
Vacancy Number: 2 Design Shifts

wages per hour: 400000 Workin 4

Figure 4appointing FTstaff

Test 4

Objective	Adding details to appoint part time staff
Action	Added details
Expected Result	Details to be added
Actual Result	Details were added
Conclusion	Test successful



Staff Hire

full time staff hire

Vacancy Number: 1 Designation: teacher JobType: FullTime

Salary: 5000 WorkingHour: 2

Vacancy Number: 1 Staff Name: niru Joining Date: 2nd april

Qualification: master Appointed By: hema

Part time staff hire

Vacancy Number: 2 Designation: technologist JobType: FullTime Shifts

wages per hour: 400000 WorkingHour: 4

Vacancy Number: 2 StaffName: kiran Joining Date: 5th april

qualification: masters Appointed By: dhiraj

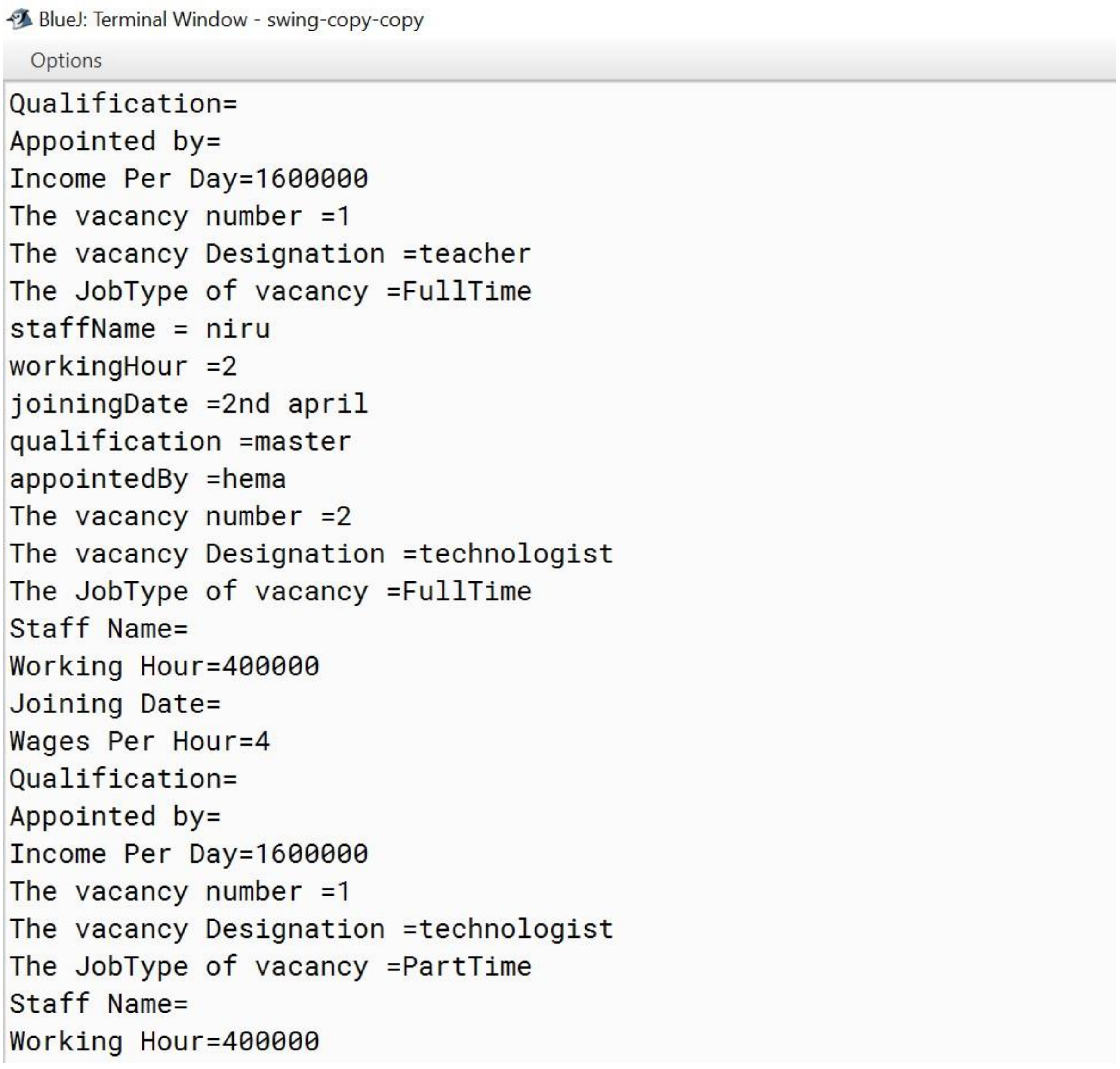
Message

Staff already hired!

Figure 5appointing part time staff

Test 5

Objective	Displaying all the details of part time staff and full time staff
Action	Added details
Expected Result	Details to be added
Actual Result	Details were added
Conclusion	Test successful


BlueJ: Terminal Window - swing-copy-copy

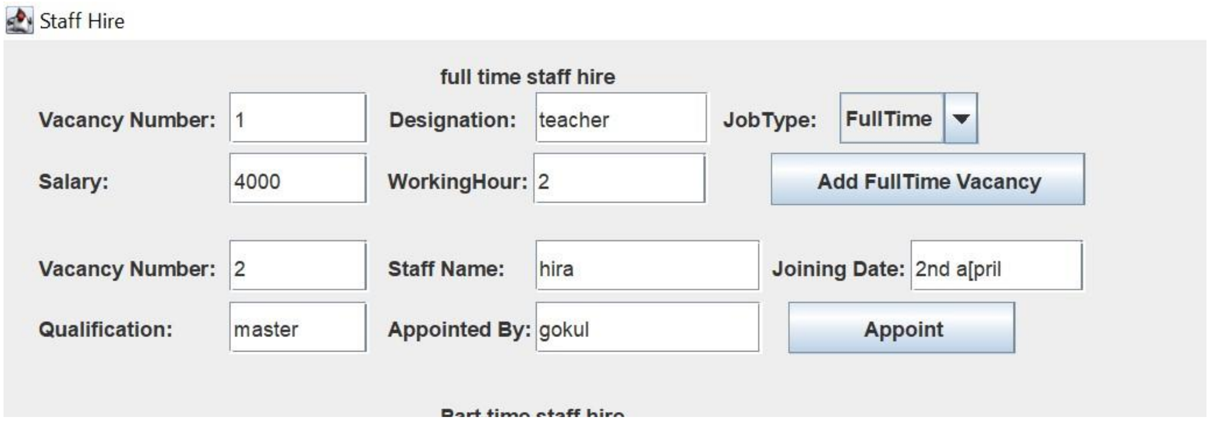
```
Qualification=
Appointed by=
Income Per Day=1600000
The vacancy number =1
The vacancy Designation =teacher
The JobType of vacancy =FullTime
staffName = niru
workingHour =2
joiningDate =2nd april
qualification =master
appointedBy =hema
The vacancy number =2
The vacancy Designation =technologist
The JobType of vacancy =FullTime
Staff Name=
Working Hour=400000
Joining Date=
Wages Per Hour=4
Qualification=
Appointed by=
Income Per Day=1600000
The vacancy number =1
The vacancy Designation =technologist
The JobType of vacancy =PartTime
Staff Name=
Working Hour=400000
```

Figure 6Displaying the test result

6.Errors

Error1

Full time Staff couldn't appointed here



Staff Hire

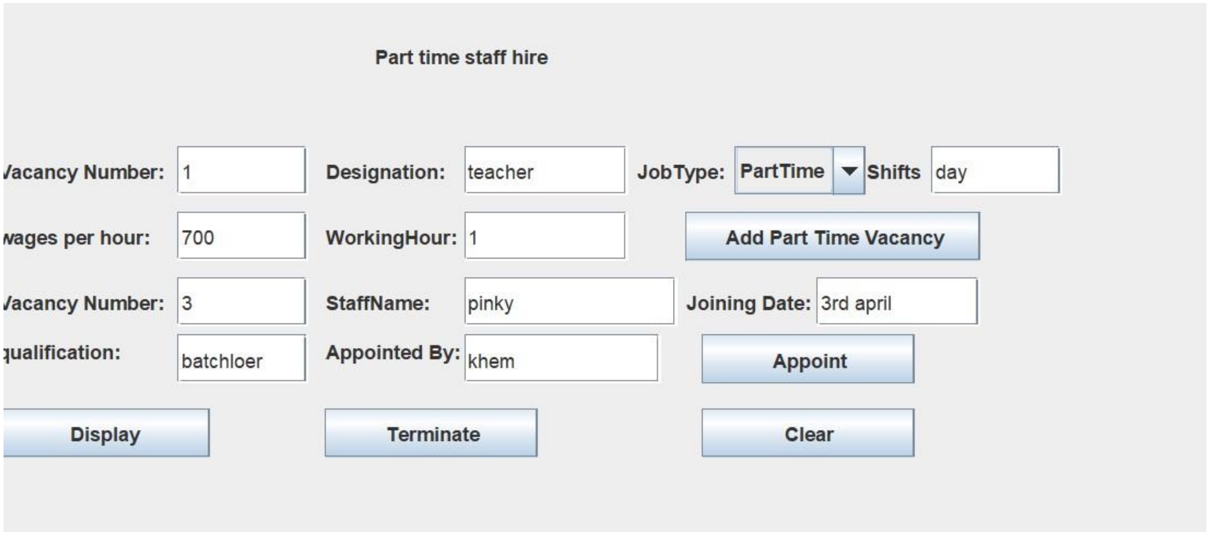
full time staff hire

Vacancy Number:	1	Designation:	teacher	JobType:	FullTime
Salary:	4000	WorkingHour:	2		
Vacancy Number:	2	Staff Name:	hira	Joining Date:	2nd a[pril
Qualification:	master	Appointed By:	gokul		

Figure 7 Error checking for FTstaff

Error2

Part time staff couldn't get appointed



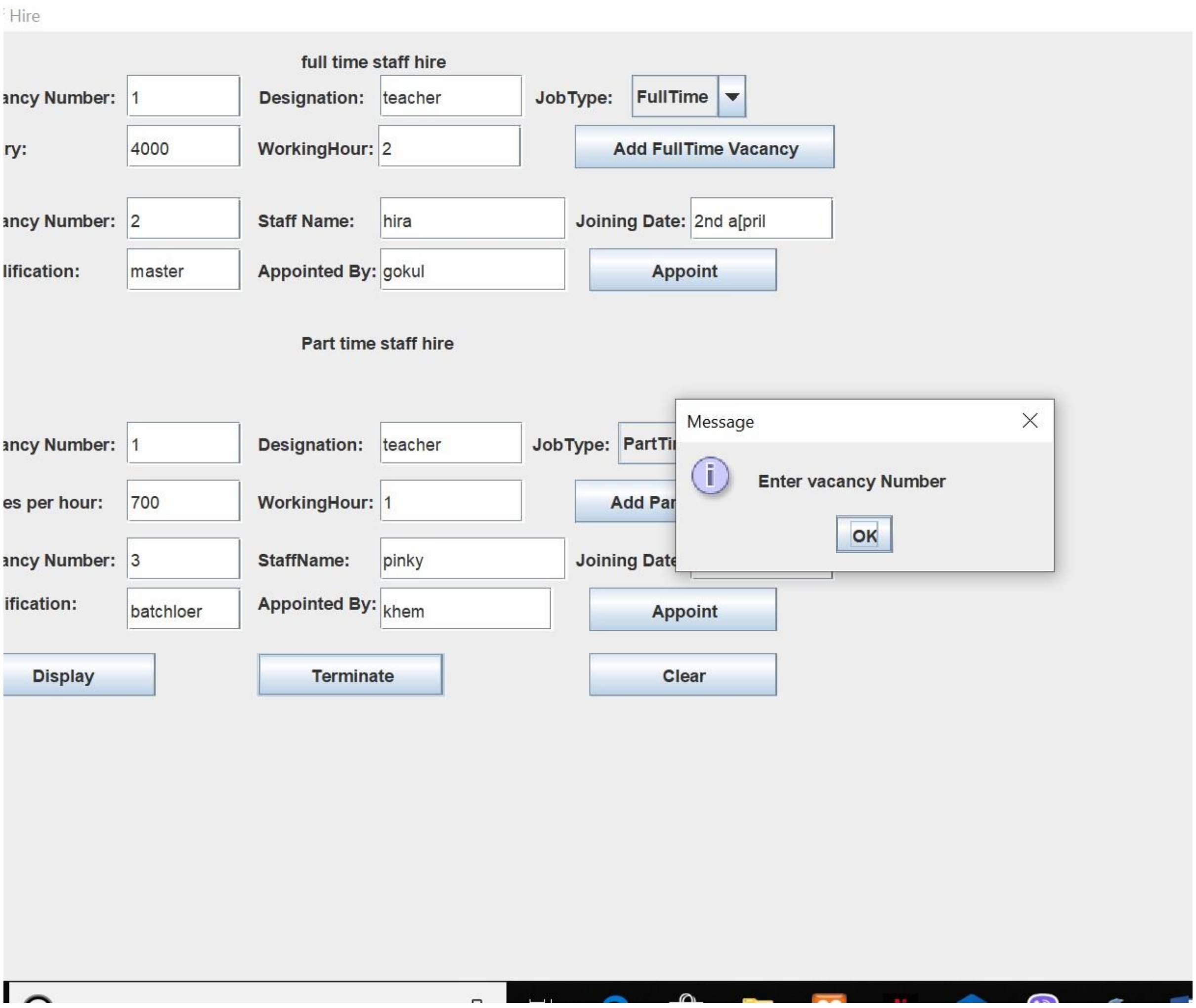
Part time staff hire

/acancy Number:	1	Designation:	teacher	JobType:	PartTime	Shifts	day
wages per hour:	700	WorkingHour:	1				
/acancy Number:	3	StaffName:	pinky	Joining Date:	3rd april		
ualification:	batchloer	Appointed By:	khem				

Figure 8Error cheking for part time stFF

Error3

Both full time and part time Staff couldn't get terminated.


Hire

full time staff hire

ancy Number: 1
Designation: teacher
JobType: FullTime
ry: 4000
WorkingHour: 2
Add FullTime Vacancy

ancy Number: 2
Staff Name: hira
Joining Date: 2nd a[pril
lification: master
Appointed By: gokul
Appoint

Part time staff hire

ancy Number: 1
Designation: teacher
JobType: PartTi
es per hour: 700
WorkingHour: 1
Add Par
ancy Number: 3
StaffName: pinky
Joining Date
ification: batchloer
Appointed By: khem
Appoint

Display
Terminate
Clear

Message
Enter vacancy Number
OK

Figure 9Error checking for terminate button

7.Conclusion.

In conclusion this course work was all about GUI. During the course work I had to went through many run time errors. It was really hard to solve those run time errors but with the help of seniors and the teachers I was finally able to solve those problems.

8.References. (arnold, 2010)

Bibliography
arnold, 2010. *gui.* www.techopedia.com/definition/5435 ed. s.l.:www.techopedia.com/definition/5435.

7.APPENDIX

Apendix1

```
/**
        * Write a description of class INGNepal here.
        *
        * @author (your name)Himanshu pandey
        * @version ( fiday 5th june 2020)
        */
       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,lblVacancyNumber1,
lblDesignaion1,lblJobType1, lblSalary1,

lblWorkingHour1,lblVacancyNumberApnt1,lblshifts,lblStaffName1,lblJoinDate1,lbl
Qualification1,lblApointedBy1;

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

txtStaffName,txtJoinDate,txtQualification,txtApointedBy,txtVacancyNumber1,
txtDesignation1,txtJobTyp1e, txtSalary1,
               txtWorkingHour1,txtVacancyNumberApnt1,
               txtStaffName1,txtJoinDate1,txtQualification1,txtApointedBy1,txtshifts;

            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(221,30,130,30);
               frm.add(lblDesignation);

               txtDesignation=new JTextField();
```

```
txtDesignation.setBounds(305,30,100,30);
frm.add(txtDesignation);

lblJobType=new JLabel("JobType:");
lblJobType.setBounds(412,30,80,30);
frm.add(lblJobType);

String jobType[]={"FullTime","PartTime"};
cmbJobType=new JComboBox(jobType);
cmbJobType.setBounds(479,30,80,30);
frm.add(cmbJobType);

lblSalary=new JLabel("Salary:");
lblSalary.setBounds(20,65,121,30);
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(304,65,100,30);
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()==btnAppointFT1)
     {
         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,appointedBy1);
                    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();
    }
 }
```

Apendix.2

Module Code and Module Title CS4001NI Programming

Assignment weightage and Type 30% Individual Coursework

Year and semester

2019-20 Autumn class 1st semester.

Student Name: Himanshu Pandey.

London Met ID: 19031311

College ID: NP01NT4A190131

Assignment Due Date: 13th January, 2020.

Assignment Submission Date: 13th January, 2020.

I confirm that I understand my course work and needed to be submitted online via Google classroom under relevant module page before deadline in order for my assignment to be accepted and marked. I am fully aware that late submission will be treated as non-submission and marks of zero will be awarded.

Table of Contents

Introduction

Its so interesting , exiting and callenging to start and work on any assignment or coursework. On 11 Dec 2019,our coursework was realesed,which was about implementing the concept of classes and objects. In this given coursework,we we have written 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 these all, StaffHire is the parent class or the main class whereas FullTimeStaffHire and PartTimeStaffHire are child class or derived class. Following coursework helps us to hire staff according to requirements to test the details of the workforce and the fame of the field workers..The main intention of this assignment is to learn 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 Diagrams

Staff Hire
□ Vacancynumber : int □ Designation : String □ Job Type : String
□ getVacancynumber() : int

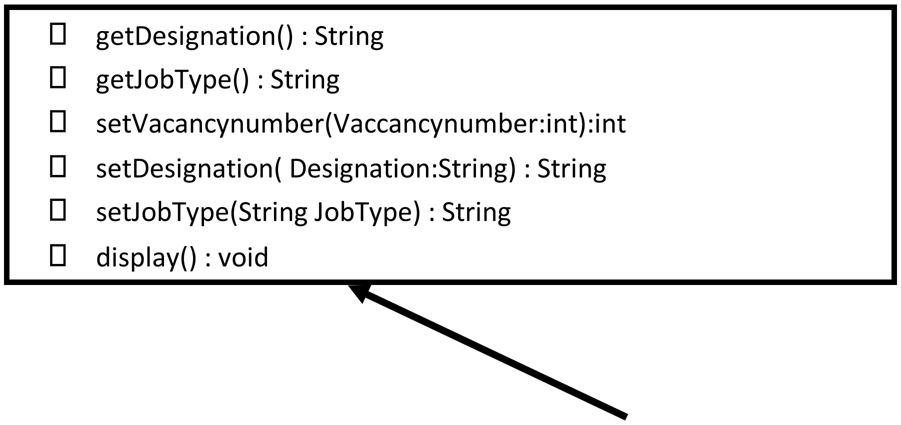
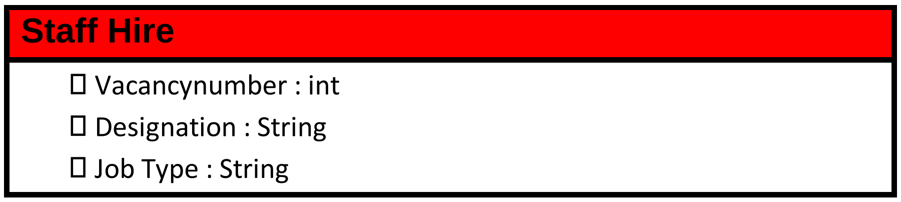
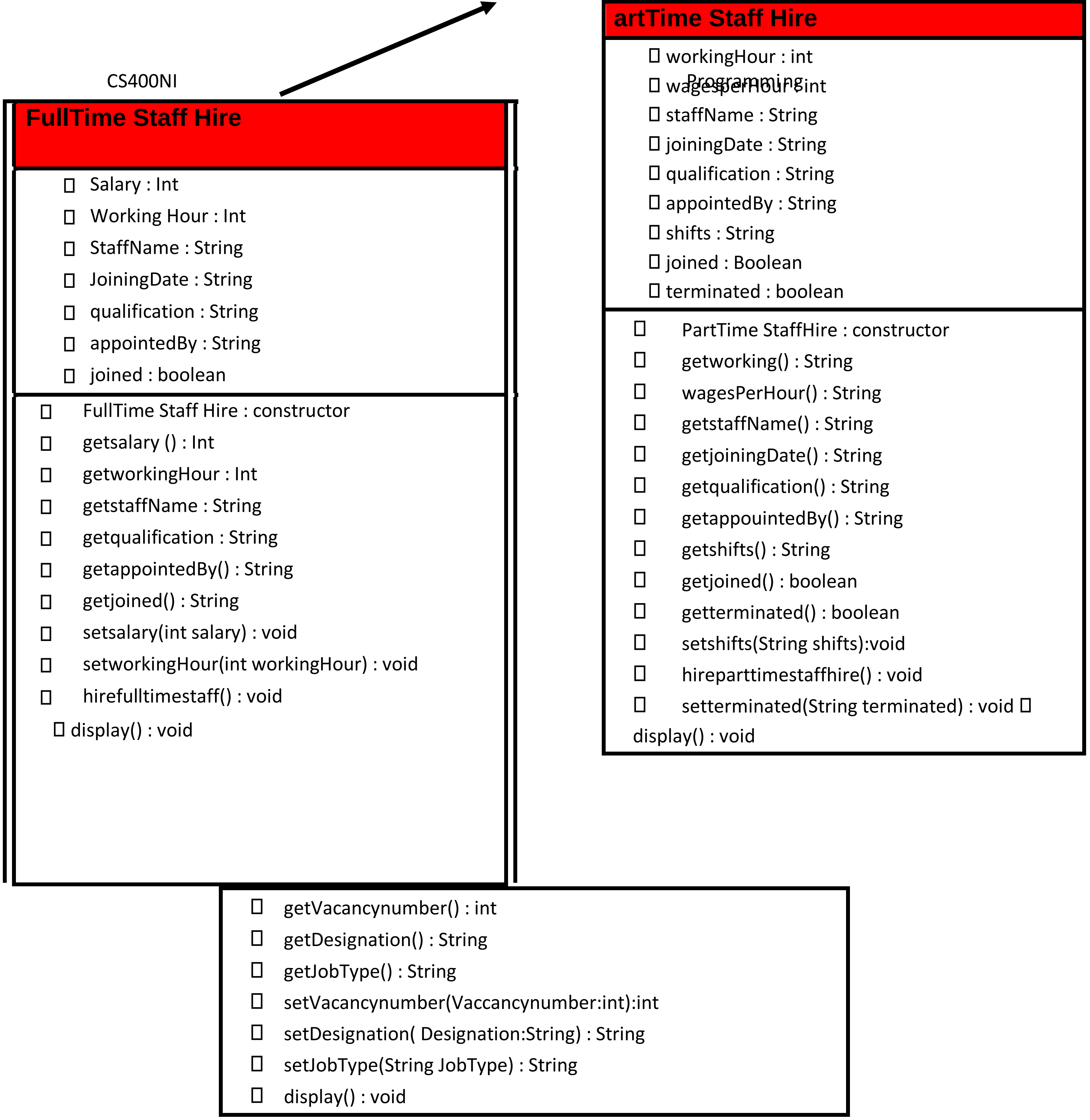

getDesignation() : String
getJobType() : String
setVacancynumber(Vaccancynumber:int):int
setDesignation(Designation:String) : String
setJobType(String JobType) : String
display() : void

Figure 1Class Diagram

StaffHire class Diagram





FullTime Staff Hire

- Salary : Int
- Working Hour : Int
- StaffName : String
- JoiningDate : String
- qualification : String
- appointedBy : String
- joined : boolean

- FullTime Staff Hire : constructor
- getsalary () : Int
- getworkingHour : Int
- getstaffName : String
- getqualification : String
- getappointedBy() : String
- getjoined() : String
- setsalary(int salary) : void
- setworkingHour(int workingHour) : void
- hirefulltimestaff() : void
- display() : void

artTime Staff Hire

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

- PartTime StaffHire : 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

- getVacancynumber() : int
- getDesignation() : String
- getJobType() : String
- setVacancynumber(Vaccancynumber:int):int
- setDesignation(Designation:String) : String
- setJobType(String JobType) : String
- display() : void

Figure 2 staffHire class diagram

FullTimeStaffHire class Diagram

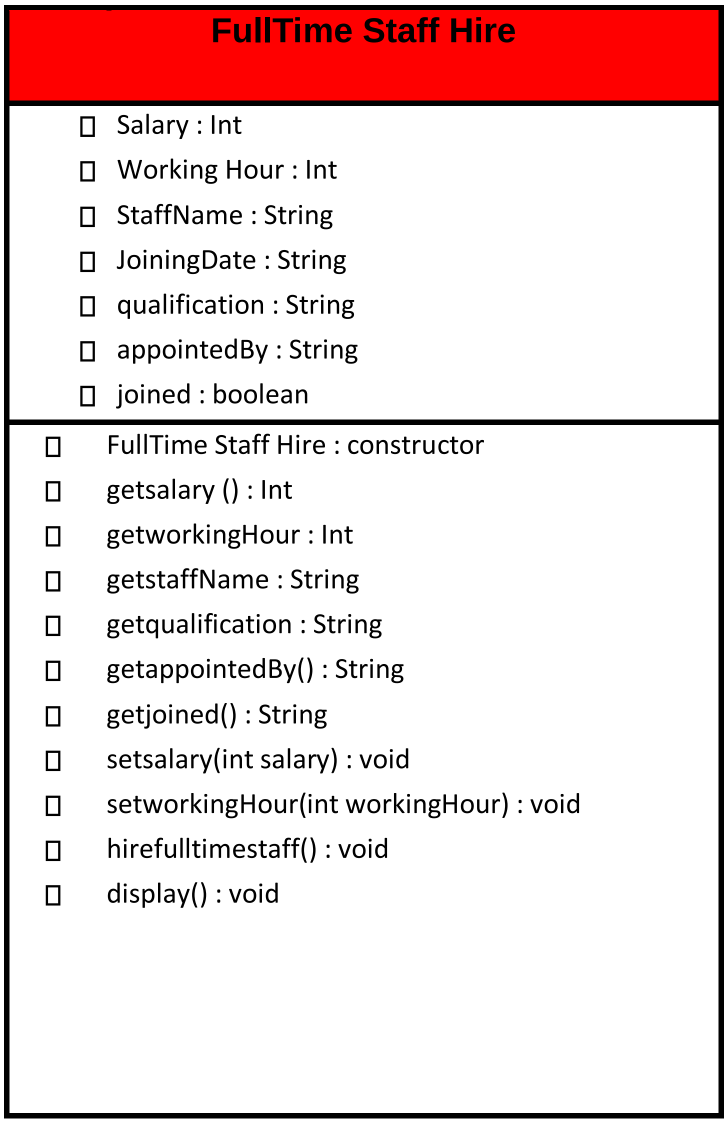
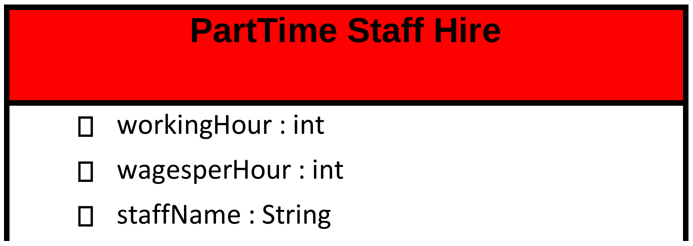
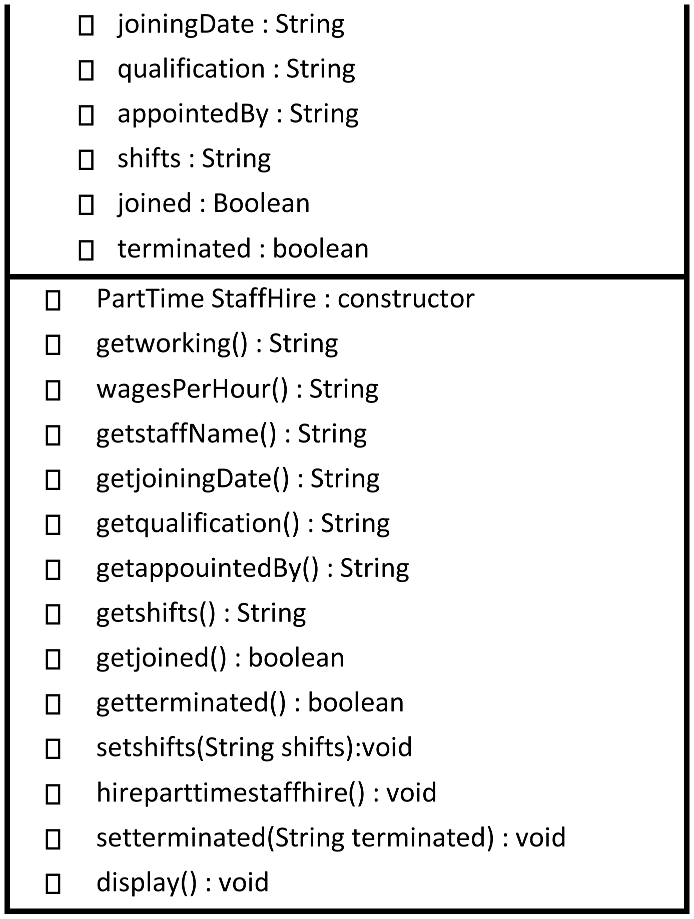


Figure 3f FullTimeStaffHire class diagram

PartTime StaffHire class Diagram





- joiningDate : String
- qualification : String
- appointedBy : String
- shifts : String
- joined : Boolean
- terminated : boolean

- PartTime StaffHire : 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 4PartTimeStaffHire class diagram

Pseudo Code Psuedo code for StaffHire

```
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
```

Psuedo code for FullTimeStaffHire

```
CREATE : FullTimeStaffHire  EXTENDS StaffHire DECLARE class
variable:
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=staffNam

e;

        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
```

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)
      DISPLAY staffName
              DISPLAY workingHour
      DISPLAY joiningDate
      DISPLAY wagesPerHour
      DISPLAY qualification
      DISPLAY appointedBy
            DISPLAY"Income Per Day="+(wagesPerHour*workingHour);
          END IF
    END DO
```

Method Description

Method Description for StaffHire class

1. **Method Name:** setVacancynumber(int Vacancynumber)

This keyword "set" method main theme is to help to access method name of every object of staffHire classs and thes keyword here is used to refer current object.

2. **Method Name:** setDesignation(String Designation)

It is also another method which has has set infront of method Name. The main purpose of this process is to access method name of every object. And the following keyword is used to refer current object.

3. **Method Name:**setJobType(String JobType)

similarly, it is also another method which contain set keyword and access method name of every object. And this keyword here is used to refer current object.

4. **Method Name:**getVacancynumber()

This keyword get infront of method name represents having return type int. and is used for accessing vacancynumber in class.

5. **Method Name:** getDesignation()

It is also another method which contains return type as String which returns string value. The following method is used for accessing Designation of eeach object throughout class.

6. **Method Name:** getJobType()

The keyword get infront of method name contains return type as String which returns String type value and this method is for accessing JobType of every object in class.

7. **Method Name:** display()

it is a method which has void as a return type and is created to print all the output of object in a class like as Vacancynumber, Designation and JobType.

Method Description for FullTimeStaffHire class

1. **Method Name:** getsalary()

This keyword "get" in front of method name has return type like int which can access int type value and returns salary.

2. **Method Name:** getworkingHour()

This keyword Consists int as return type which access int value and returns workingHour.

3. **Method Name:** getstaffName()

The keyword get contains String as return type which can returns String value as staffname.

4. **Method Name:** getjoiningDate()

The following method have String as return type and it returns String value as joiningDate.

5. **Method Name:** getqualification()

It is also another type of method have keyword get infront of method nameand it has String as return type which returns qualification in String form.

6. **Method Name:** getappointedBy()

This method has get keyword. This method has String as return type and it returns which return appointedBy in String form.

7. **Method Name:** getjoined()

This method is for joined which has Boolean as return type. This method was make to check the joining status which can checks the joined or not joined staff status and returns a message including true or false.

8. **Method Name:** setsalary(int salary)

This method has “set” keyword which represents it is a mutator method. This method was created to access method name of every object of FullTimestaffhire class. It has int as return type.

9. **Method Name:** setworkingHour(int workingHour)

The main purpose of creating this method is to access every object in FullTimeHire class. It contains int as return type.

10. **Method Name:** hirefulltimestaff(String staffName, String joiningDate,String qualification, String appointedBy).This method was created to check whether the staff has joined or not which is Boolean as return type andchecks joining status and print out the appropriate message as per condition like true or false.

11. **Method Name:** display()

This is also a mutator method which has void as a return type. This method is created to print all the output of object in a class. Such as staffName, workingHour, joiningDate, qualification and appointedBy.

Method Description for PartTimeStaffHire class

1. **Method Name:** getworkingHour()

It contains int as return type which access int value and returns workingHour.

2. **Method Name:**getwagesPerHour()

It contains int as return type which access int value and returns wagesPerHour.

3. **Method Name:** getstaffName()

This method has String as return type which returns String value as staffname.

4. **Method Name:** getjoiningDate()

It has String as return type which indeed returns String value as joiningDate.

5. **Method Name:** getqualification()

This method has string as return type which returns String value in qualification.

6. **Method Name:** getappointedBy()

This method is previous with get keyword. This method has String as return type which returns which return appointedBy in String form.

7. **Method Name:** getshifts()

Similarly,
This method has String as return type which returns string value in shifts like mornig, day or night.

8. **Method Name:** getjoined()

This method is for joined which has Boolean as return type. This method was created to check the joining status which checks the joined or not joined staff status and returns a message including true or false.

9. **Method Name:** getterminated()

This method is for terminated which has Boolean as return type. This method was created to check the terminated status. It checks the terminated status of staff and returns a message including true or false.

10. **Method Name:** setshifts(String shifts)

This method also has set keyword which represents it also as mutator method. Its main purpose of creating this method is to access every object in PartTimeHire class.
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. It was created to check if the staff has joined or not and return Boolean as return type. It checks joining status and print out the appropriate message .

12. **Method Name:** void setterminated()

This method also has set keyword . It has void as return type. The main purpose of creating this method is to access every object in PartTimeHire class. this gives the output as per condition.

13. **Method Name:** void display()

This is also another method which has void as a return type and is is created to print all the output of object in a class. Such as staffName, workingHour, joiningDate, Wages Per Hour, qualification, appointedBy and Income Per Day.

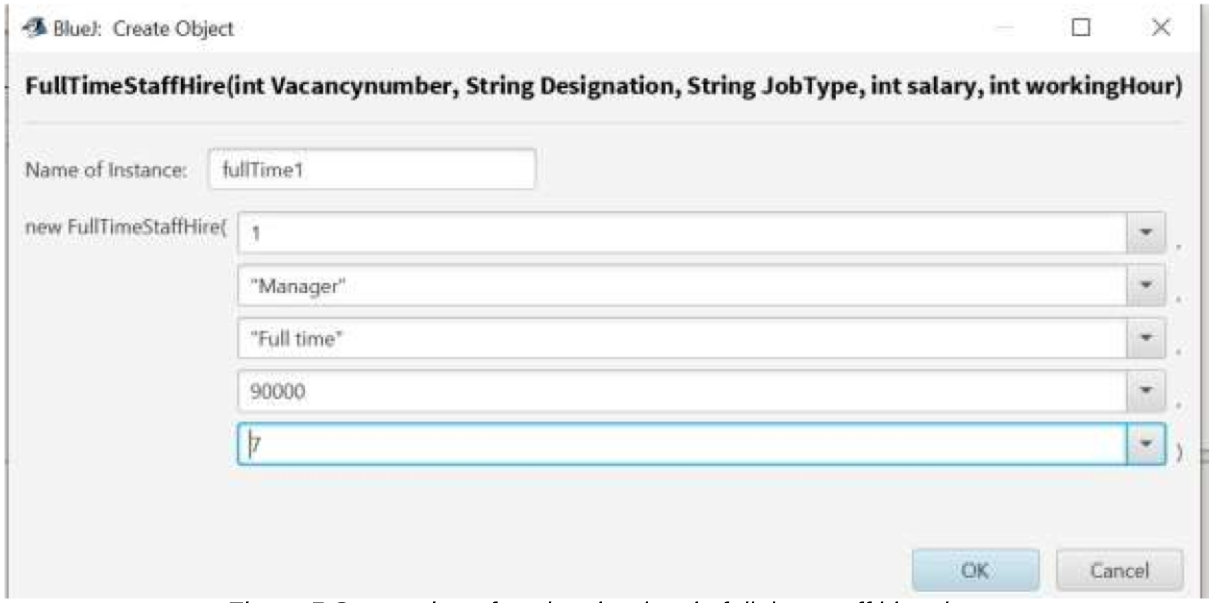
Testing(Inspection)

Test 1- 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:	➢ The FullTimeStaffHire is called with the following arguments: Vacancynumber:1 Designation: “Manager” JobType: “FULL TIME” salary:90000 workingHour:7 ➢ Inspection of the FullTImeStaffHire. ➢ void hirefulltimestaff(String staffName, String joiningDate, String qualification, String appointedBy) satffName:”Himanshu pandey” joiningDate:”2019/10/01” qualification:”Master” appointedBy:”C.E.O” ➢ 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.

Table 1: TEST NUMBER 1

<u>OUTPUT RESULT:</u>

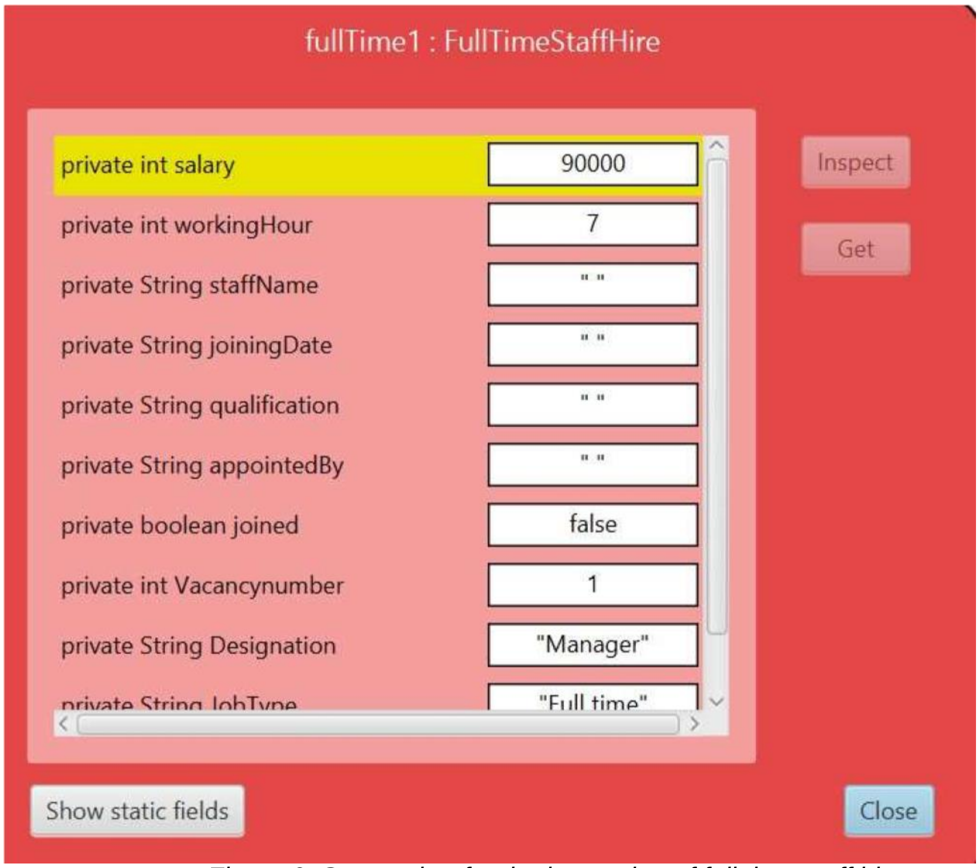

BlueJ: Create Object

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

Name of Instance: fullTime1

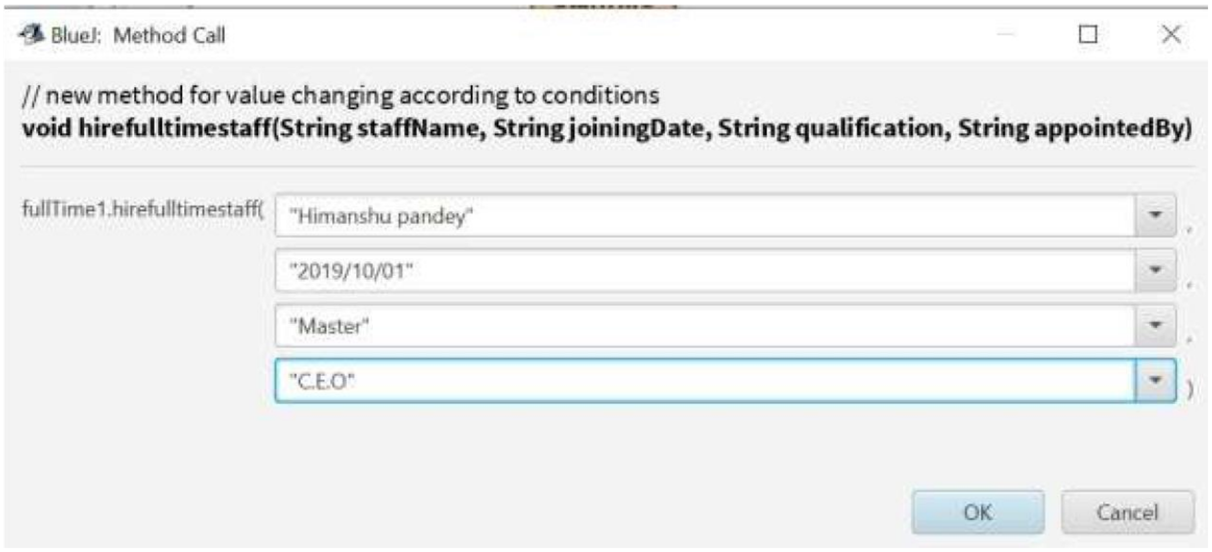
new FullTimeStaffHire(1 , "Manager" , "Full time" , 90000 , 7)

Figure 5:Screenshot of assign the data in full time staff hire class


fullTime1 : FullTimeStaffHire

private int salary	90000
private int workingHour	7
private String staffName	" "
private String joiningDate	" "
private String qualification	" "
private String appointedBy	" "
private boolean joined	false
private int Vacancynumber	1
private String Designation	"Manager"
private String JobType	"Full time"

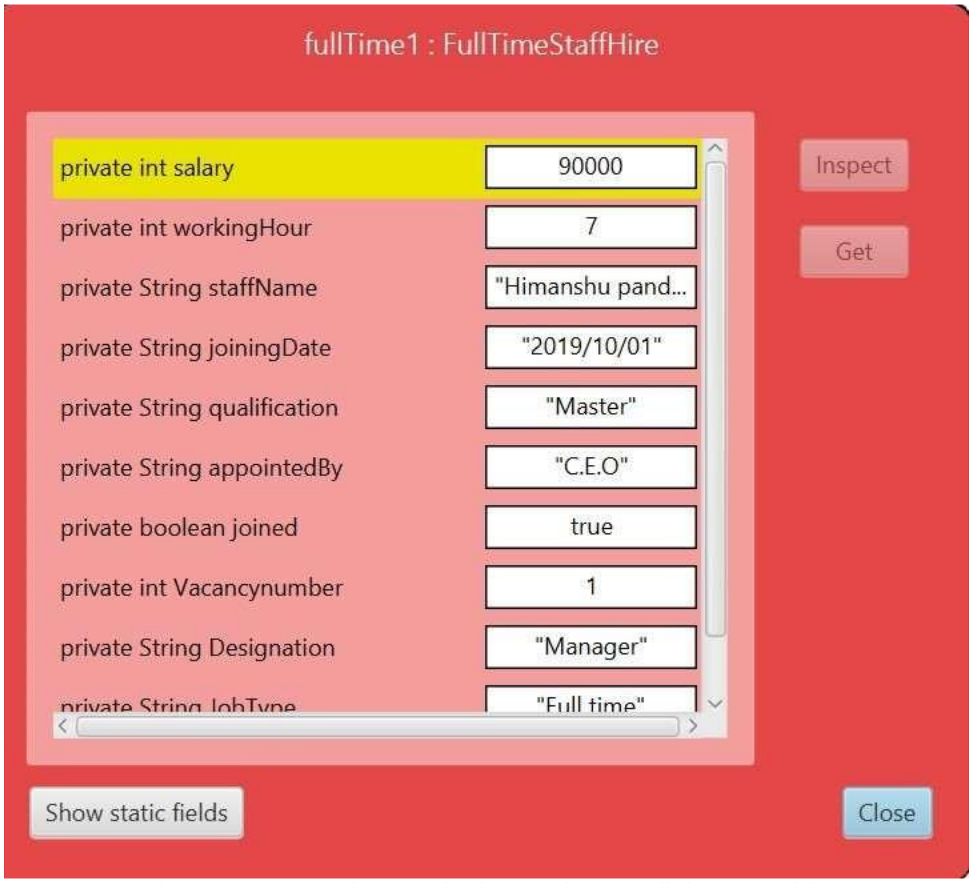
Figure 6: Screenshot for the inspection of full time staff hire



// new method for value changing according to conditions
void hirefulltimestaff(String staffName, String joiningDate, String qualification, String appointedBy)

fullTime1.hirefulltimestaff("Himanshu pandey", "2019/10/01", "Master", "C.E.O")

Figure 7: Screenshot of assign the data in full time staff hire class



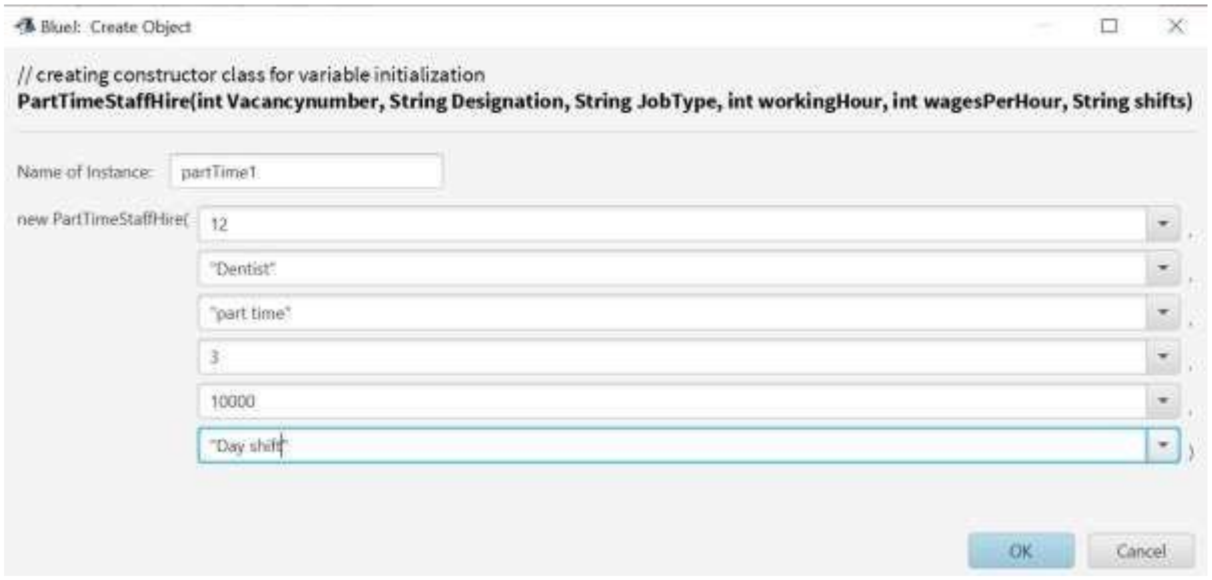
fullTime1 : FullTimeStaffHire

private int salary	90000
private int workingHour	7
private String staffName	"Himanshu pand...
private String joiningDate	"2019/10/01"
private String qualification	"Master"
private String appointedBy	"C.E.O"
private boolean joined	true
private int Vacancynumber	1
private String Designation	"Manager"
private String JobType	"Full time"

Figure 8: Screenshot for the inspection of full time staff hire

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

TEST NO:	2
Objective:	To PARTTimeStaffHire and re-inspect the PARTTimeStaff class
Action:	➢ The PARTStaffHire is called with the following arguments: Vacancynumber:12 Designation: “Dentist” JobType: “PART TIME” salary:10000 workingHour:3 shifts: ”DAY SHIFT” ➢ Inspection of the partTImeStaffHire. ➢ void hirefulltimestaff(String staffName, String joiningDate, String qualification, String appointedBy) satffName:”Niraj Pandey” joiningDate:”20/10/16” qualification:”Master ” appointedBy:”C.E.O ➢ 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.

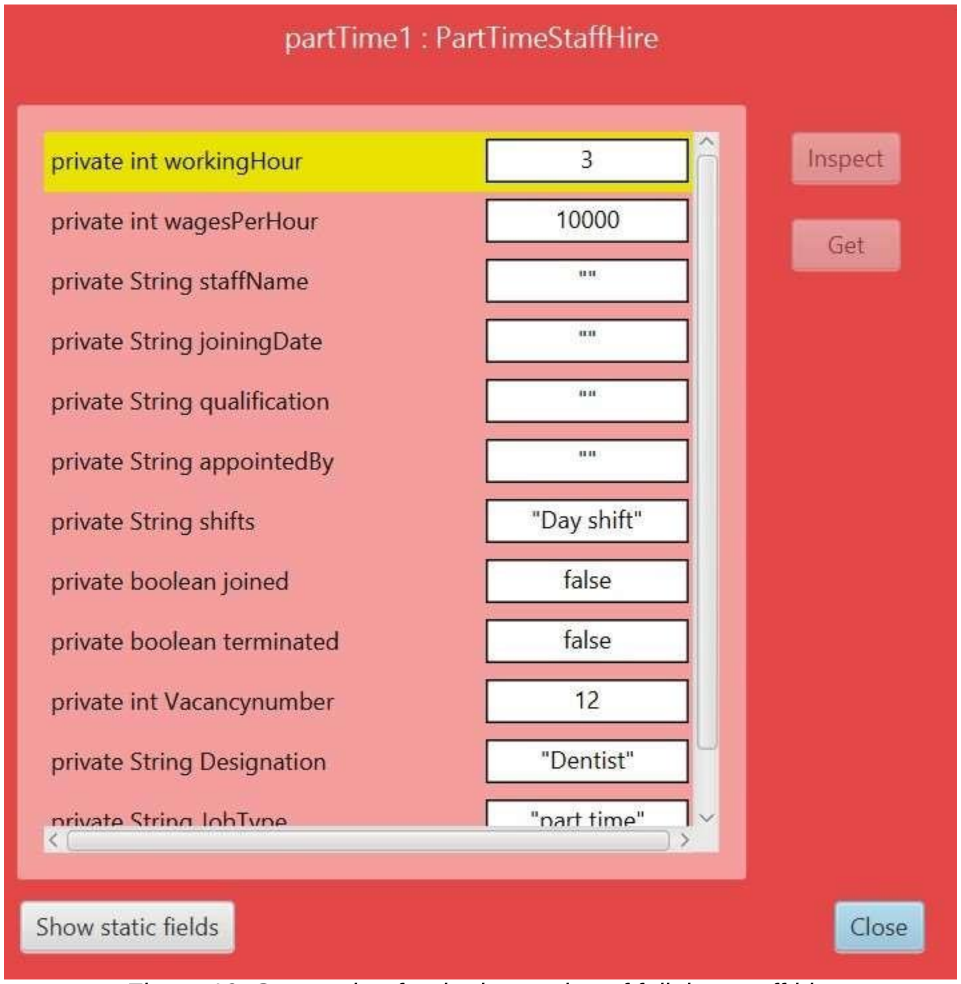


// creating constructor class for variable initialization
PartTimeStaffHire(int Vacancynumber, String Designation, String JobType, int workingHour, int wagesPerHour, String shifts)

Name of Instance: partTime1

new PartTimeStaffHire(12 , "Dentist" , "part time" , 3 , 10000 , "Day shift")

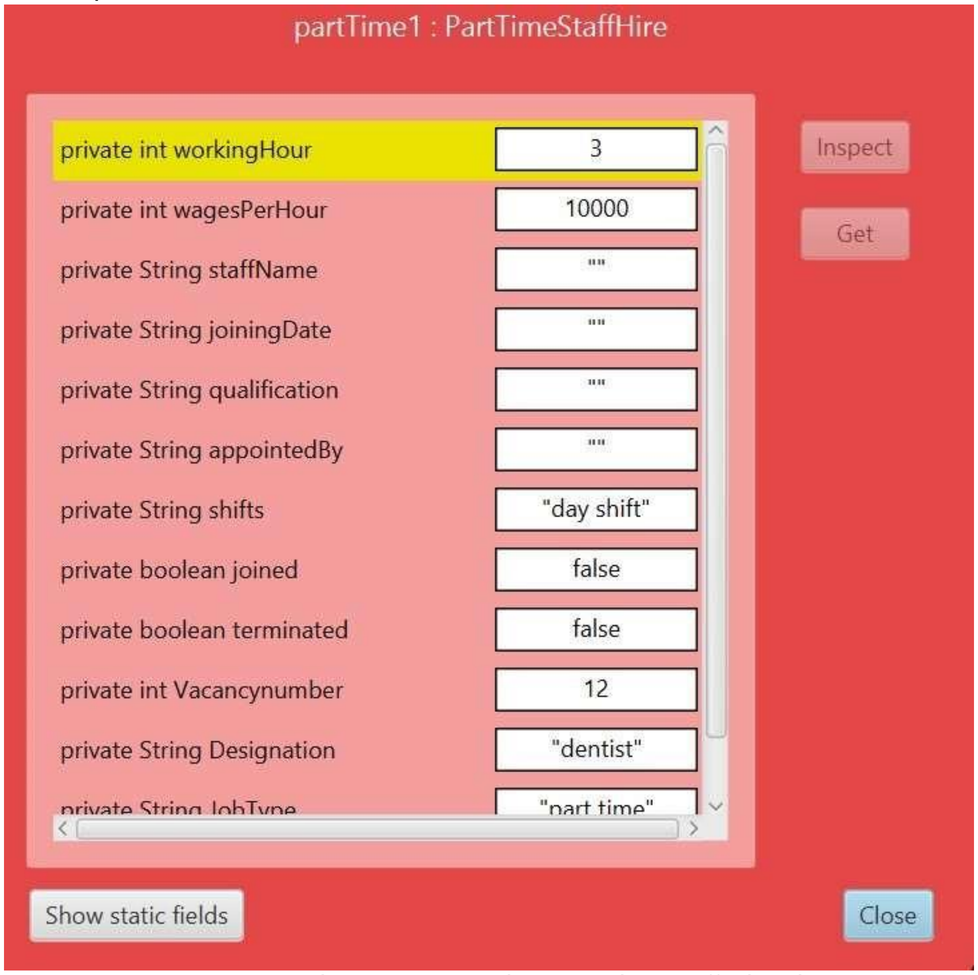
Figure 9: Screenshot of assign the data in part time staff hire class



private int workingHour	3
private int wagesPerHour	10000
private String staffName	""
private String joiningDate	""
private String qualification	""
private String appointedBy	""
private String shifts	"Day shift"
private boolean joined	false
private boolean terminated	false
private int Vacancynumber	12
private String Designation	"Dentist"
private String JobType	"part time"

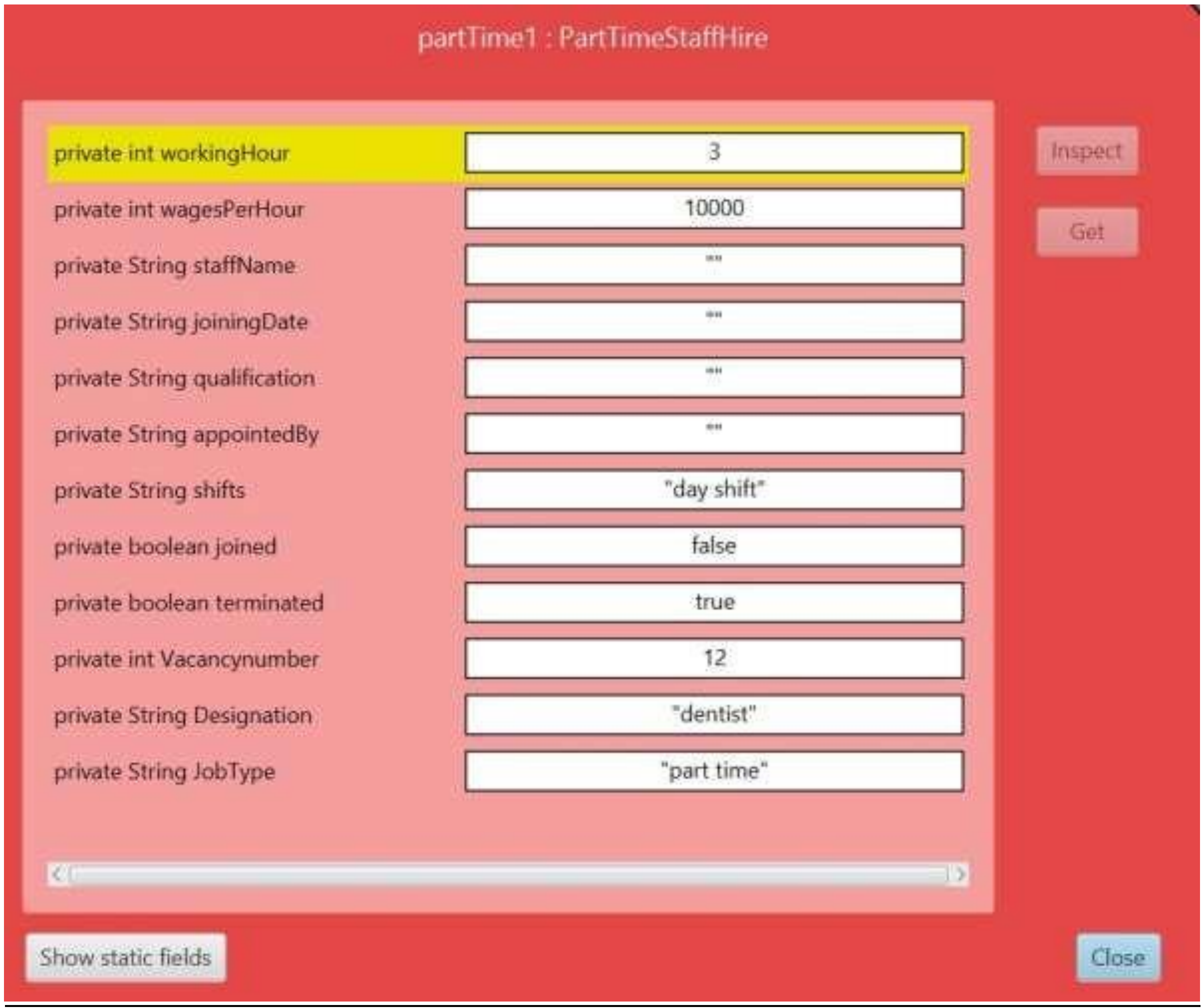
Figure 10: Screenshot for the inspection of full time staff hire

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


partTime1 : PartTimeStaffHire

private int workingHour	3
private int wagesPerHour	10000
private String staffName	""
private String joiningDate	""
private String qualification	""
private String appointedBy	""
private String shifts	"day shift"
private boolean joined	false
private boolean terminated	false
private int Vacancynumber	12
private String Designation	"dentist"
private String JobType	"part time"

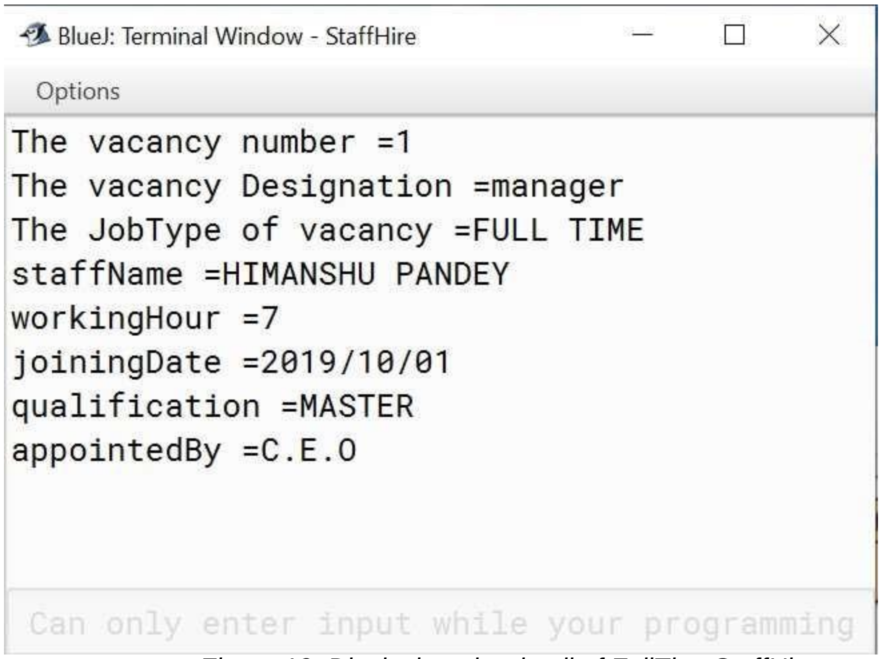
Figure 11INspecting part Time staff Hire class


partTime1 : PartTimeStaffHire

private int workingHour	3
private int wagesPerHour	10000
private String staffName	""
private String joiningDate	""
private String qualification	""
private String appointedBy	""
private String shifts	"day shift"
private boolean joined	false
private boolean terminated	true
private int Vacancynumber	12
private String Designation	"dentist"
private String JobType	"part time"

Figure 12 Re-inspecting partTime staffHire class

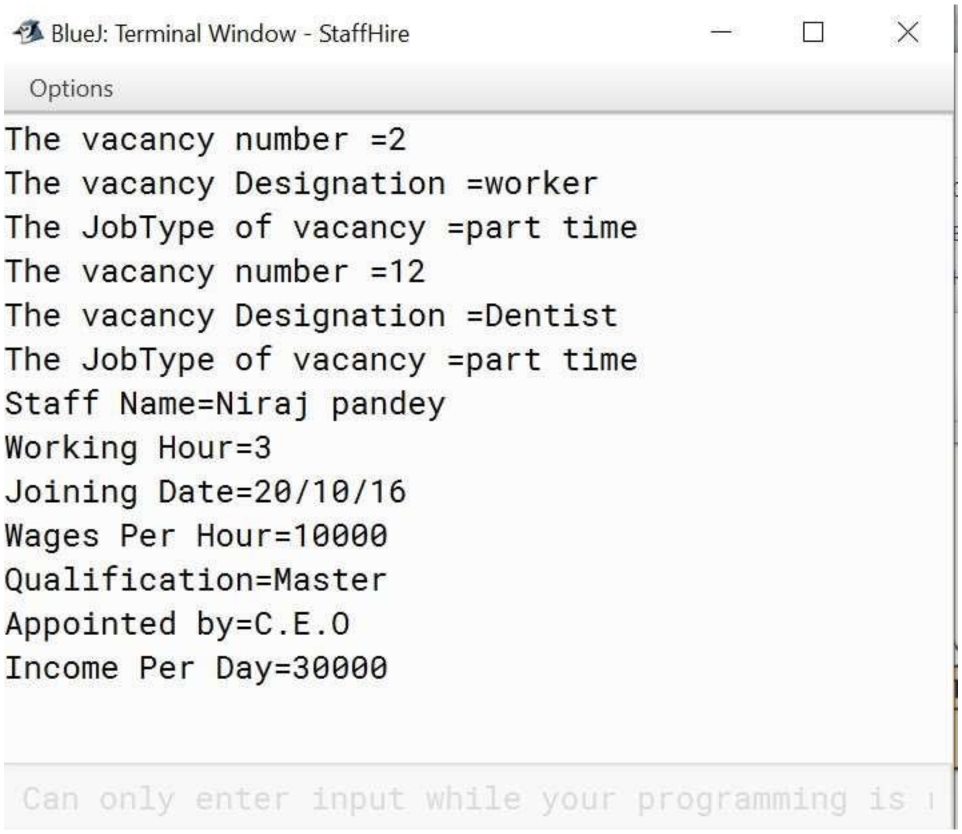
TEST 4: Display the detail of FullTimeStaffHire and PartTimeStaffHire Class.


BlueJ: Terminal Window - StaffHire
Options

```
The vacancy number =1
The vacancy Designation =manager
The JobType of vacancy =FULL TIME
staffName =HIMANSHU PANDEY
workingHour =7
joiningDate =2019/10/01
qualification =MASTER
appointedBy =C.E.O
```

Can only enter input while your programming

Figure 13: Displaying the detail of FullTimeStaffHire



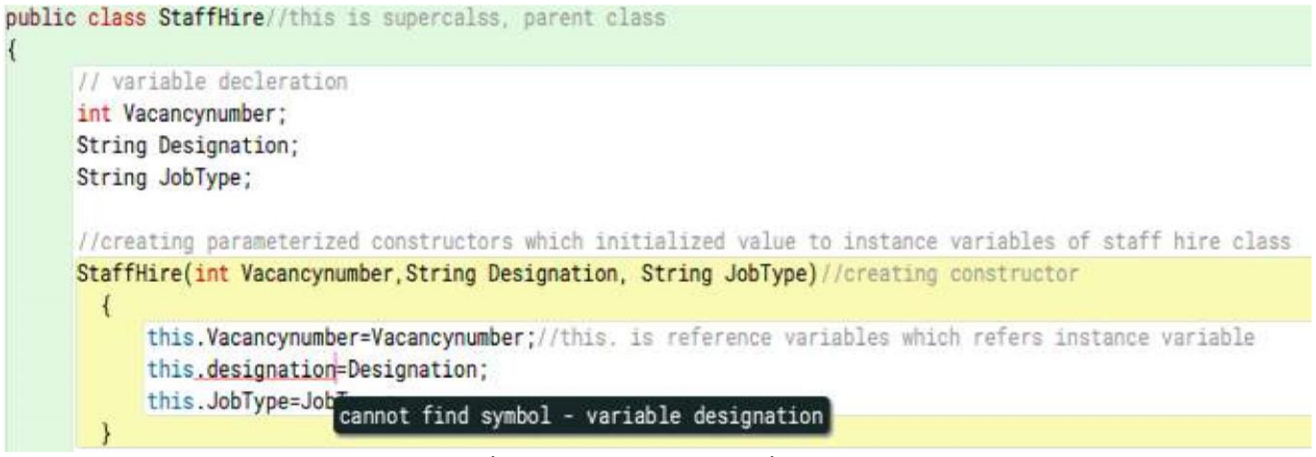
```
The vacancy number =2
The vacancy Designation =worker
The JobType of vacancy =part time
The vacancy number =12
The vacancy Designation =Dentist
The JobType of vacancy =part time
Staff Name=Niraj pandey
Working Hour=3
Joining Date=20/10/16
Wages Per Hour=10000
Qualification=Master
Appointed by=C.E.O
Income Per Day=30000
```

```
Can only enter input while your programming is 
```

Figure 14: Displaying the detail of PartTimeStaffHire

Error Detection And Correction

Error 1 :

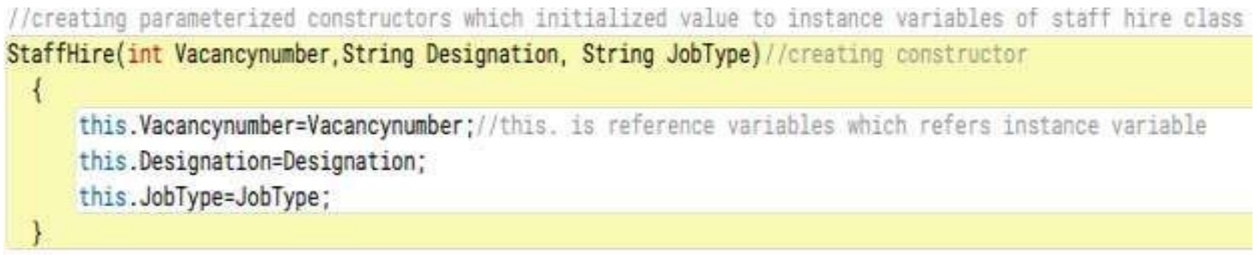


```
public class StaffHire//this is supercalss, parent class
{
        // variable decleration
        int Vacancynumber;
        String Designation;
        String JobType;

        //creating parameterized constructors which initialized value to instance variables of staff hire class
        StaffHire(int Vacancynumber,String Designation, String JobType)//creating constructor
         {
              this.Vacancynumber=Vacancynumber;//this. is reference variables which refers instance variable
              this.designation=Designation;
              this.JobType=Job
         }
```

```
cannot find symbol - variable designation
```

Figure 15 Error Detected

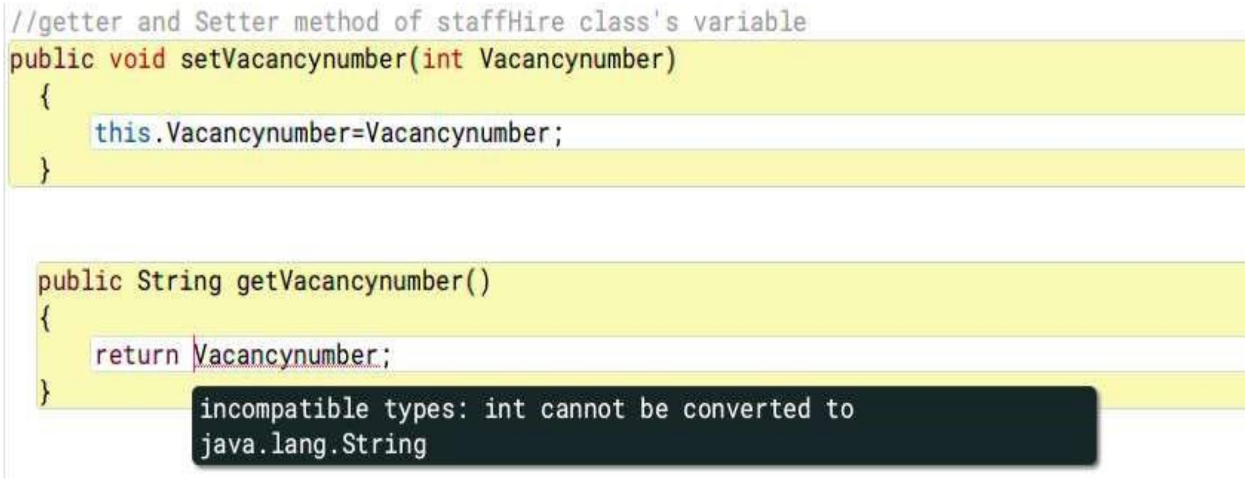


```
//creating parameterized constructors which initialized value to instance variables of staff hire class
StaffHire(int Vacancynumber,String Designation, String JobType)//creating constructor
  {
      this.Vacancynumber=Vacancynumber;//this. is reference variables which refers instance variable
      this.Designation=Designation;
      this.JobType=JobType;
  }
```

Figure 16Error corrected

This error was caused by variable. The variable name in instance variable was not similar as declared variables. So to fix this I’ve made the variable name similar in instance variable.

Error 2 :



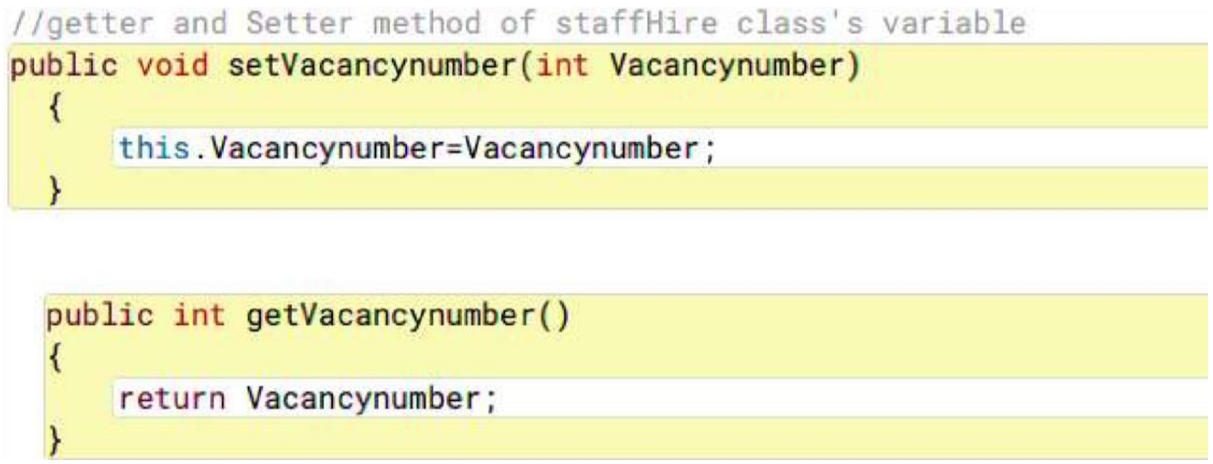
```
//getter and Setter method of staffHire class's variable
public void setVacancynumber(int Vacancynumber)
  {
      this.Vacancynumber=Vacancynumber;
  }

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

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

Figure 17Error Detected

Error 2 Correction:



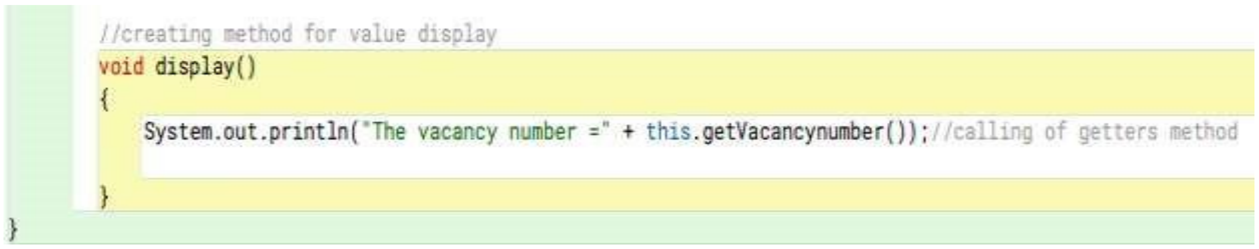
```
//getter and Setter method of staffHire class's variable
public void setVacancynumber(int Vacancynumber)
  {
       this.Vacancynumber=Vacancynumber;
  }


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

Figure 18Error corrected

The error here I have encountered was cause by loosy conversion of int type variable to string. So, to overcome this error I have made the return type same as previous variable as above while declaring variables. I have made both return type to int.

Error 3 :

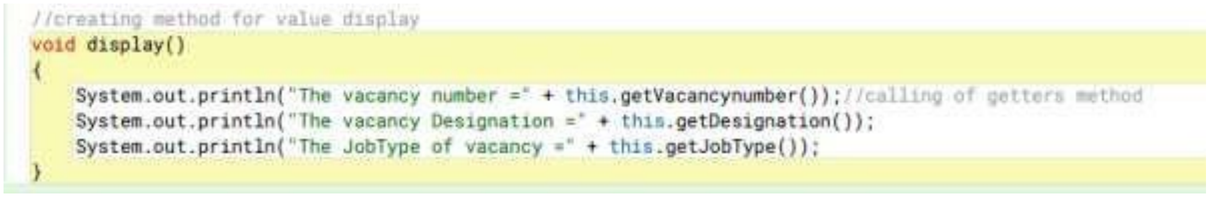


```
    //creating method for value display
    void display()
    {
        System.out.println("The vacancy number =" + this.getVacancynumber());//calling of getters method

    }
}
```

Figure 19 Error 3 Detected

Error 3 :



```
//creating method for value display
void display()
{
    System.out.println("The vacancy number =" + this.getVacancynumber());//calling of getters method
    System.out.println("The vacancy Designation =" + this.getDesignation());
    System.out.println("The JobType of vacancy =" + this.getJobType());
}
```

Figure 20Error corrected

This error was not getting the required output. So, to fix this error made some addition in display method. I have added a designation and JobType as new display method.

Conclusion

AS far as we know that, it’s quite difficult to complete the whole assignment at single time or single try. To get easier we need to work on it continuously. I’ve read all the given question and scenarios properly after leasing of our assignments. Somehow it was so difficult and quite confusing at initially. However, to get rid out of this problem, I research on various topics like concept of inheritance, classes and objects. However, I’ve gained some useful and fruitful knowledge for stating this assignment and then after I have started to write code for my StaffHire.

Initially, writing code for StaffHire was not that much difficult for me.But when I started my code for FullTimeStaffHire and PartTimeStaffHire it was so frustrating and hard for me at the same time. So to eradicate and overcome such difficulties I have asked my module teachers , my tutors as well as my friends to solve such diificulties which I have been facing earlier. Timely and relevantly, I am able to complete the coding part .

To analyze my work. I headed towards testing the program that I have developed. I inspected all the kind and again I re-inspected all the guidelines and corrected them accordingly. Overall this coursework was full of chalanging as well as interesting at the sametime.I’ve learned clearly about oop(Object Oriented programming) from this
given assignment

Appendix

StaffHire class

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

```
    String JobType;

    //creating constructors
    StaffHire(int Vacancynumber,String Designation, String JobType)//creating constructor
     {
        this.Vacancynumber=Vacancynumber;//this. is reference variables which refers instance
variable        this.Designation=Designation;        this.JobType=JobType;
     }

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



     public int getVacancynumber()
     {
        return Vacancynumber;
     }


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


     public String getDesignation()
     {
        return Designation;
     }


    public void setJobType(String JobType)
     {
        this.JobType="JobType";
     }
    public String getJobType()
    {
      return JobType;
    }

    //creating method for value display
void display()
    {
      System.out.println("The vacancy number =" +
this.getVacancynumber());//calling of getters method
      System.out.println("The vacancy Designation =" + this.getDesignation());
      System.out.println("The JobType of vacancy =" + this.getJobType());
    }
}
```

Fulltime StaffHire

```
class FullTimeStaffHire extends StaffHire // loading of super class in sub calss or parent class in child
class

{
//instance variable
decleration    int salary;    int
workingHour;    String
staffName;
  String joiningDate;
  String qualification;
String appointedBy;
boolean joined;

   //creating constructor class
  FullTimeStaffHire (int Vacancynumber,String Designation,String JobType,int salary,int workingHour)
  {
      super(Vacancynumber,Designation,JobType);//calling superclass 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 value changing according to condition*/     public void setsalary(int salary)
     {
        this.salary=salary;
if (joined=true)
        System.out.println("You have been appointed Therefore, not possible to change" +salary);


     }


   public void setworkingHour(int workingHour)
     {
       this.workingHour=workingHour;
       System.out.println("The working Hour is" + workingHour);
     }

     /**new method for value changing according to conditions*/          public void
hirefulltimestaff(String staffName,String joiningDate,String qualification,String appointedBy)

    {

       if(joined==true)
       {
          System.out.println(" staffName" + staffName);
          System.out.println("joiningDate" +joiningDate);


       }
else
       {
         this.staffName=staffName;
this.joiningDate=joiningDate;          this.qualification=qualification;
this.appointedBy=appointedBy;          joined=true;
       }
    }
    //display method for displaying attributes value
public void display()
    {
      super.display();
if(joined==true)
      {
      System.out.println("staffName =" +this.getstaffName());//calling of getters method
      System.out.println("workingHour =" +this.getworkingHour());
      System.out.println("joiningDate =" +this.getjoiningDate());
      System.out.println("qualification =" +this.getqualification());
      System.out.println("appointedBy =" +this.getappointedBy());
      }
    }

}
```

Parttime StaffHire

```
public class PartTimeStaffHire extends StaffHire//creating child class of parent class and loading
parent class with extends  keyword
{
  //variable
decleration    int
workingHour;    int
wagesPerHour;
String staffName;
  String joiningDate;
  String qualification;
  String
appointedBy;
String shifts;
boolean joined;
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=true;
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));
       }
    }

}
```

FINISH