



Module Code & Module Title

CS4001NI Programming

Report title

Assignment Weightage & Type

30% Individual Coursework

2019/20 Autumn

Student name: Ishan Gurung

London Met ID: 19031315

College ID: NP01NT4A190139

Assignment Due date:13 January,2020.

Assignment submission Date:13 January,2020.

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 submission will be treated as non-submission and a mark of zero will be awarded.

Contents

Introduction.....	1
Class Diagram.....	2
<i>Pseudo code</i>	5
1.Pseudo code for Staff Hire class:	5
2. Pseudo code of Full Time Staff Hire:	7
3. Pseudo code of Part Time Staff Hire:	11
METHOD DESCRIPTION.....	16
Method for staff hire.....	16
Test 1- To inspect FULLTimestaffhire class, appointed the full staff and reinspect the FULLTIME STAFFHIRE CLASS.	23
Test 2: Inspect PartTimeStaffHire Class, appoint part time staff, and reinspect the PartTimeStaffHire Class.....	26
Test 3: Inspect PartTimeStaffHire Class, change the termination status of a staff, and re- inspect the PartTimeStaffHire Class	29
TEST 4: Display the detail of FullTimeStaffHire and PartTimeStaffHire Class.	31
ERRORS AND CORRECTION	32
1. Error 1: Syntax error	32
APPENDIX	36
1. STAFF HIRE.....	36
2. FULL TIME STAFF HIRE:	38
3. PART TIME STAFF HIRE	43

Figure 1:Partial class diagram	2
Figure 2 StaffHire.....	3
Figure 3 FullTimeStaffHire	3
Figure 4 PartTimeStaffHire	4
Figure 5 Screenshot of assign the data in fullTimeStaffHire class.....	24
Figure 6 Screenshot for inspecting of Full time staff Hire	24
Figure 7Screenshot of assign the data in fullTimeStaffHire class	25
Figure 8Screenshot for inspecting of Full time staff Hire	25
Figure 9 Screenshot for inspecting of Full time staff Hire	27
Figure 10Screenshot for inspecting of part time staff Hire	28
Figure 11 Screenshot for inspecting of Full time staff Hire	28
Figure 12 Screenshot for inspecting of part time staff Hire	28
Figure 13 Screenshot of inspecting part time hire	29
Figure 14 Changing termination status	30
Figure 15 Displaying the detail of Fulltime StaffHire.....	31
Figure 16 Displaying the detail of parttime staff hire.....	31
Figure 17 error 1 detection	32
Figure 18 error `1 rectified.....	32
Figure 19 error 2 detection	33
Figure 20 error 2 rectified.....	33
Figure 21 error 3 detection	34
Figure 22 error 3 rectified.....	34

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 FullTimeStaffHire and PartTimeStaffHire 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

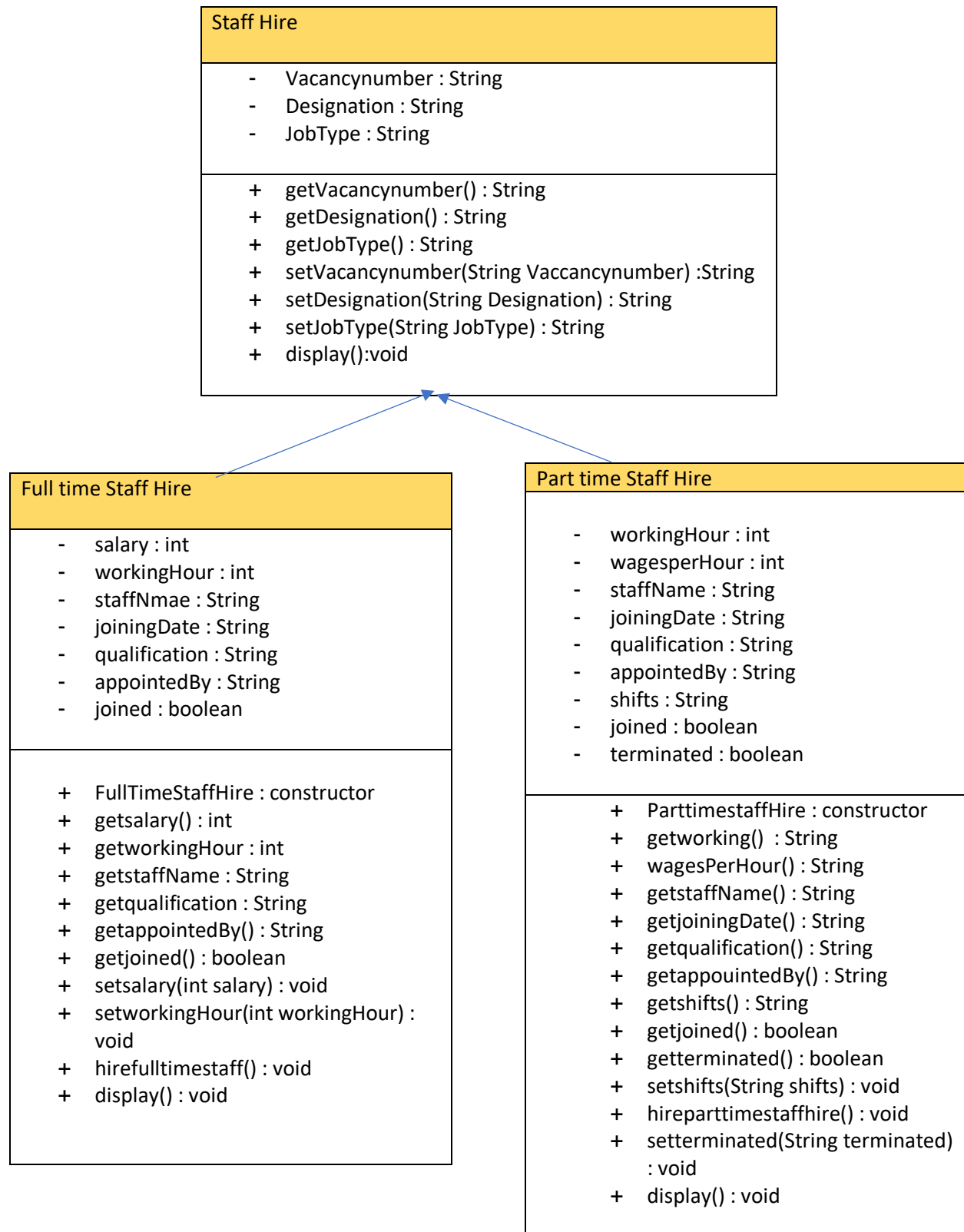


Figure 1:Partial class diagram

staff hire

Staff Hire
<ul style="list-style-type: none"> - JobType : String - Vacancynumber : String - Designation : String
<ul style="list-style-type: none"> + getVacancynumber() : String + getDesignation() : String + getJobType() : String + setVacancynumber(String Vacancynumber) :String + setDesignation(String Designation) : String + setJobType(String JobType) : String + display():void

Figure 2 StaffHire

Full time Staff Hire

Full time Staff Hire
<ul style="list-style-type: none"> - salary : int - workingHour : int - staffNmae : String - joiningDate : String - qualification : String - appointedBy : String - joined : boolean
<ul style="list-style-type: none"> + 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 3 FullTimeStaffHire

Part time Staff Hire
<ul style="list-style-type: none">- workingHour : int- wagesperHour : int- staffName : String- joiningDate : String- qualification : String- appointedBy : String- shifts : String- joined : Boolean- terminated : boolean
<ul style="list-style-type: none">+ 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 4 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 Vaccancynumber;

END DO

FUNCTION setVacancynumber(int Vaccancynumber)

DO

this.Vaccancynumber=Vaccancynumber;

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 access every 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:	<ul style="list-style-type: none"> ➤ 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:

new FullTimeStaffHire(,
"Manager" ,
"Full time" ,
 ,
)

OK Cancel

Figure 5 Screenshot of assign the data in fullTimeStaffHire class

fullTime1 : FullTimeStaffHire

private int salary	<input type="text" value="60000"/>
private int workingHour	<input type="text" value="6"/>
private String staffName	<input type="text" value=""/>
private String joiningDate	<input type="text" value=""/>
private String qualification	<input type="text" value=""/>
private String appointedBy	<input type="text" value=""/>
private boolean joined	<input type="text" value="false"/>
private int Vacancynumber	<input type="text" value="2"/>
private String Designation	<input type="text" value="Manager"/>
private String JobType	<input type="text" value="Full time"/>

Show static fields Close

Inspect Get

Figure 6 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")

OK Cancel

Figure 7 Screenshot of assign the data in fullTimeStaffHire class

fullTime1 : FullTimeStaffHire

private int salary	60000	Inspect
private int workingHour	6	
private String staffName	""	Get
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"	

Show static fields Close

Figure 8 Screenshot 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:	<ul style="list-style-type: none"> ➤ 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 PartTimeStaffHire. ➤ void hireparttimestaff(String staffName, String joiningDate, String qualification, String appointedBy) staffName:"srijan Thapa magar" joiningDate:"20/10/12" qualification:"High school" appointedBy:"Manager" ➤ Re-inspection of the PartTimeStaffHire
Expected Result:	The part time teacher would be appointed.
Actual Result:	The Part time staff was appointed.
Conclusion:	The test is successful.

+

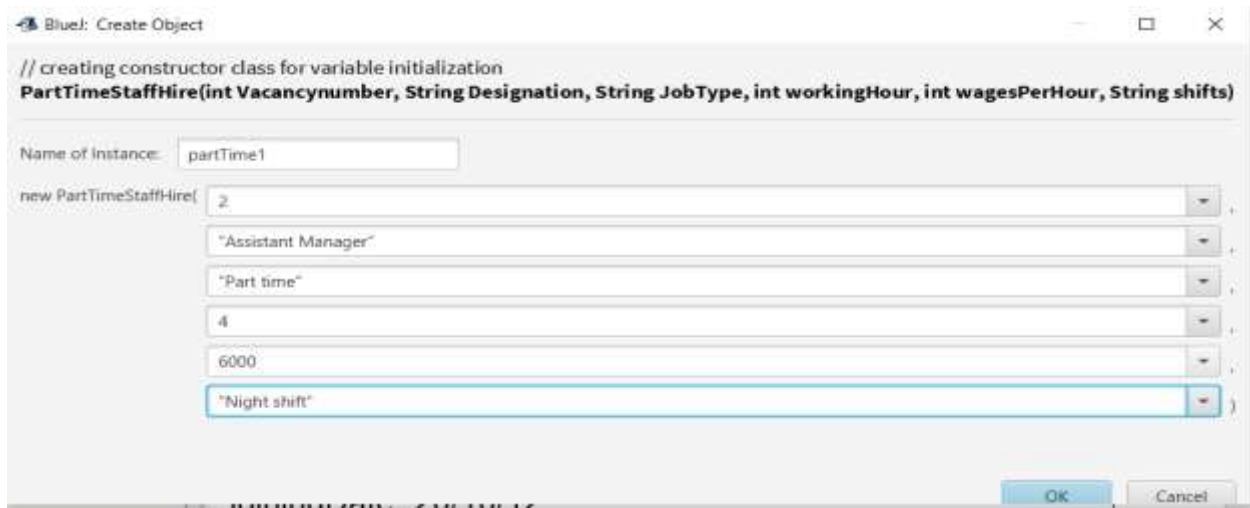


Figure 9 Screenshot for inspecting of Full time staff Hire

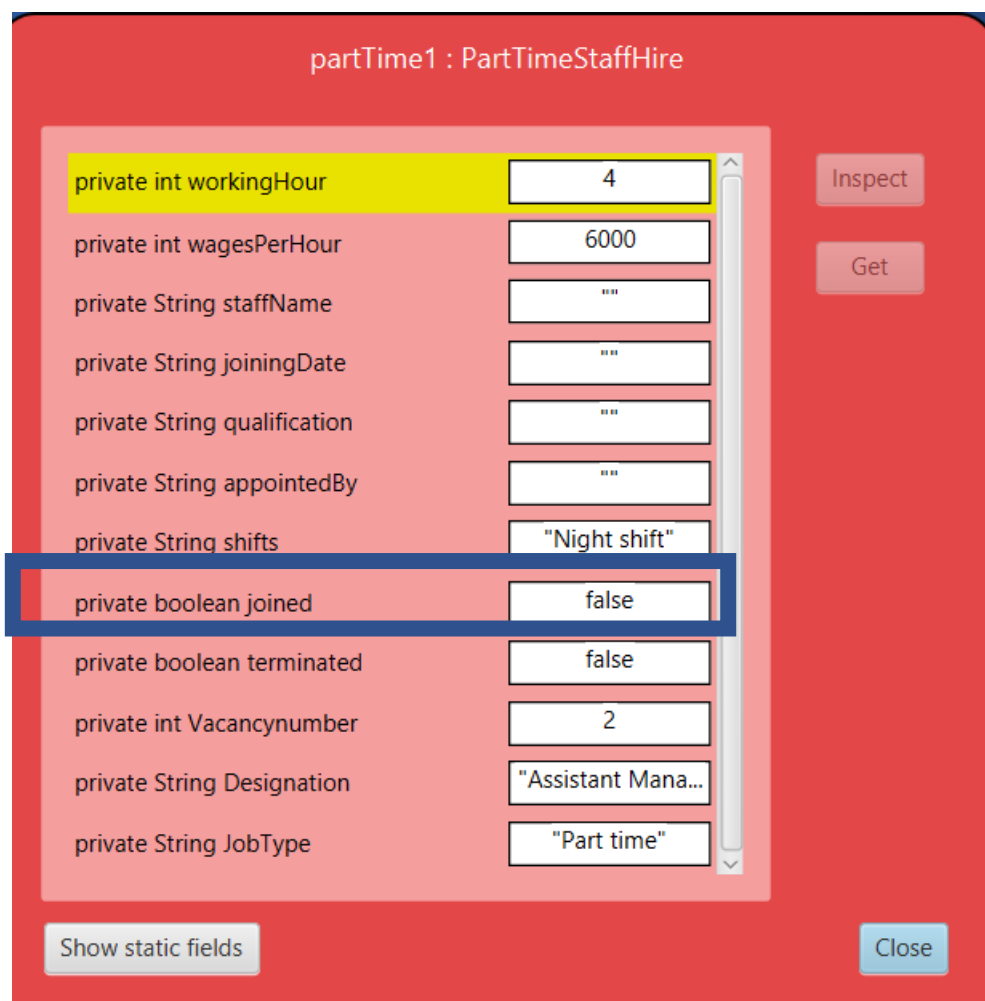


Figure 10 Screenshot for inspecting of part time staff Hire

Blue: Method Call

```
// 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")

OK Cancel

Figure 11 Screenshot for inspecting of Full time staff Hire

partTime1 : PartTimeStaffHire

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"

Inspect Get

Show static fields Close

Figure 12 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

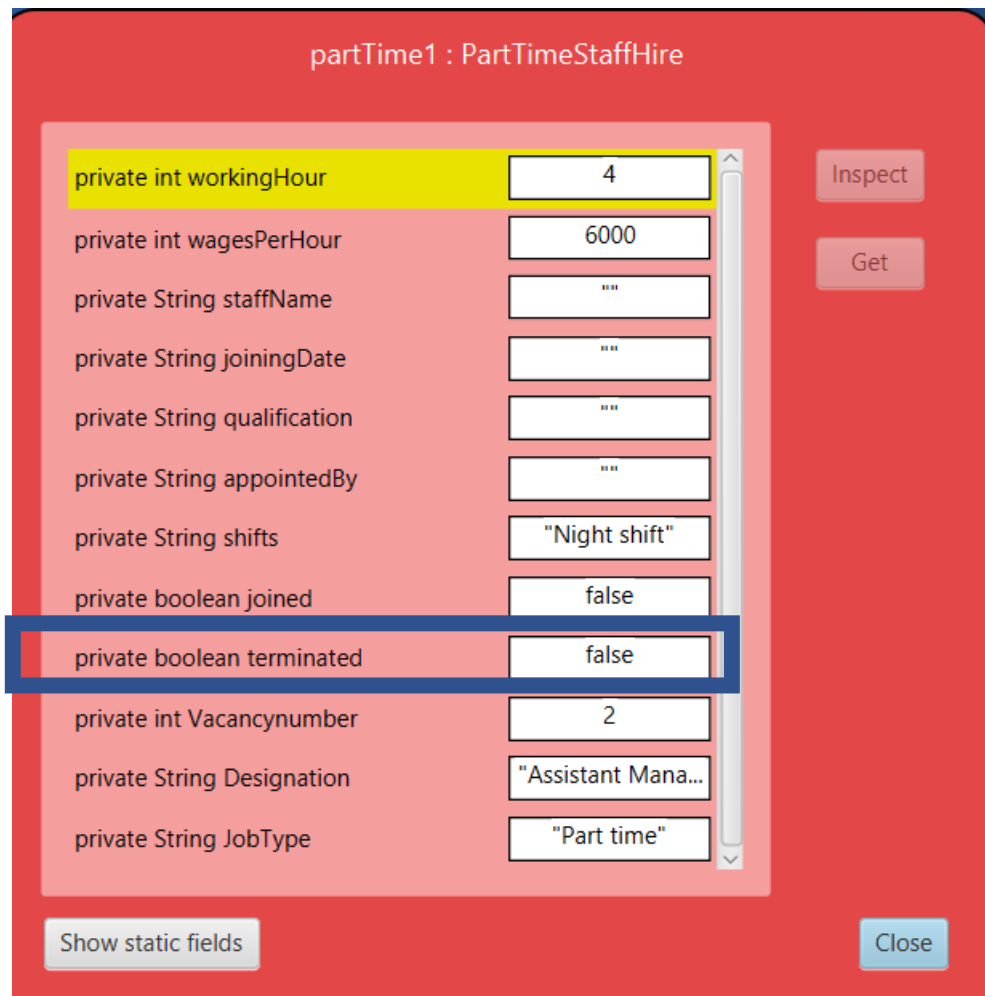


Figure 13 Screenshot of inspecting part time hire

partTime1 : PartTimeStaffHire

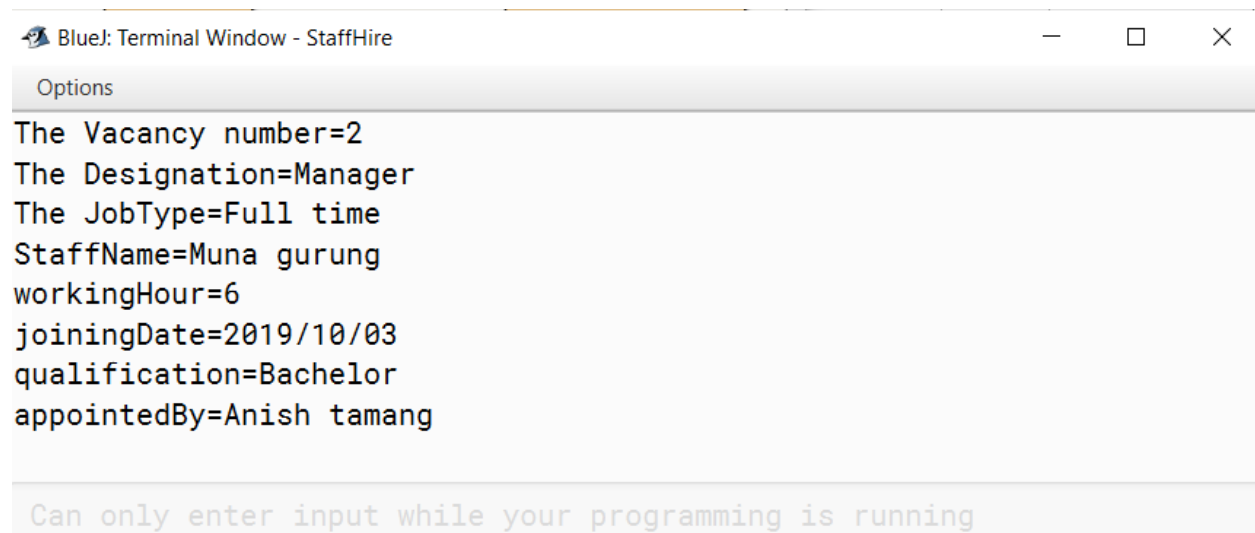
private int workingHour	4	Inspect
private int wagesPerHour	6000	
private String staffName	""	Get
private String joiningDate	""	
private String qualification	""	
private String appointedBy	""	
private String shifts	"Night shift"	
private boolean joined	false	
private boolean terminated	true	
private int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

< >

Show static fields Close

Figure 14 Changing termination status

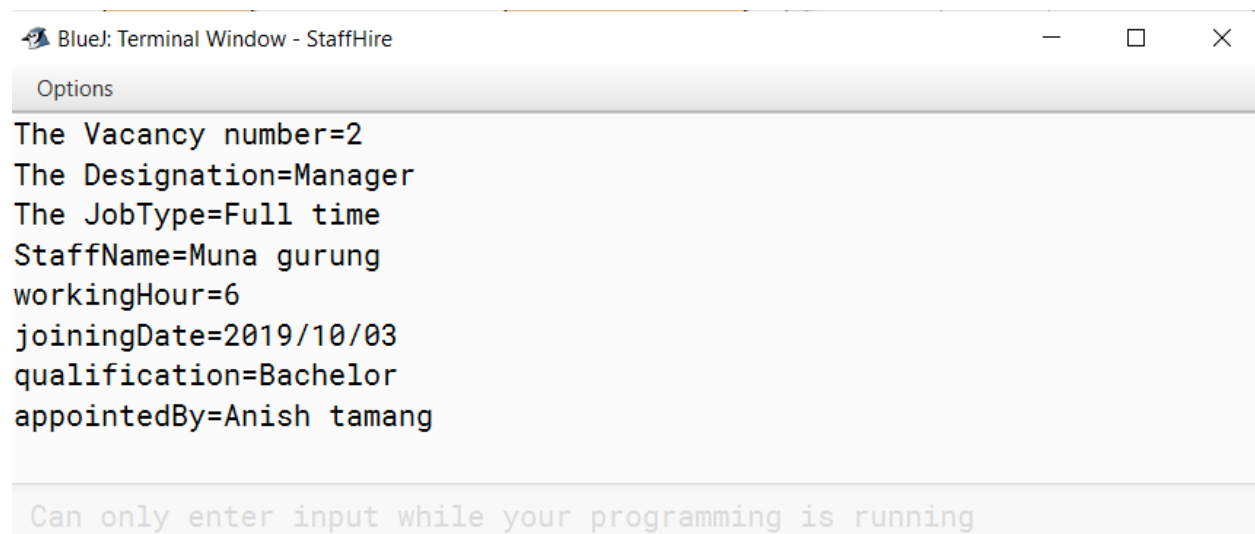
TEST 4: Display the detail of FullTimeStaffHire and PartTimeStaffHire Class.



```
BlueJ: Terminal Window - StaffHire
Options
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

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

Figure 15 Displaying the detail of Fulltime StaffHire



```
BlueJ: Terminal Window - StaffHire
Options
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

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

Figure 16 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 declaration
    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 17 error 1 detection

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

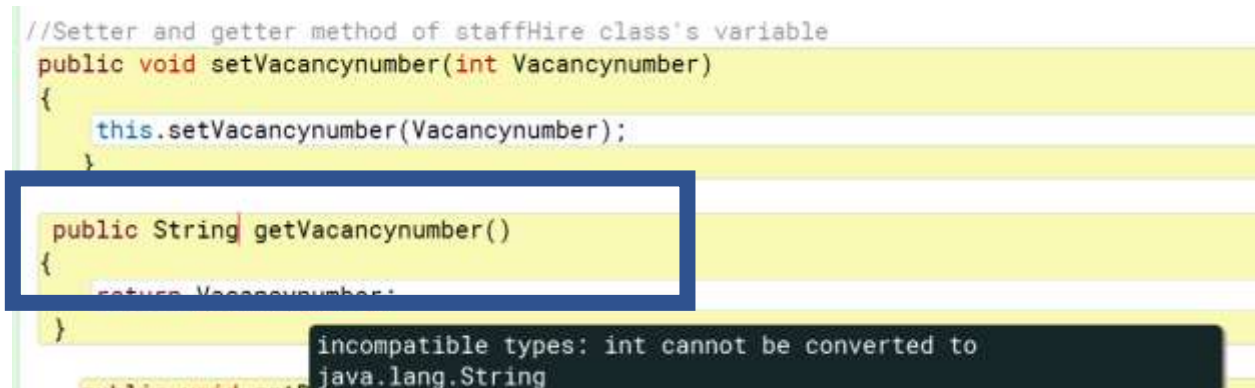
    // creating the constructor class
    StaffHire(int Vacancynumber,String Designation, String JobType)
    {
        this.Vacancynumber=Vacancynumber;
        this.Designation=Designation;
        this.JobType=JobType;
    }
}
```

Figure 18 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

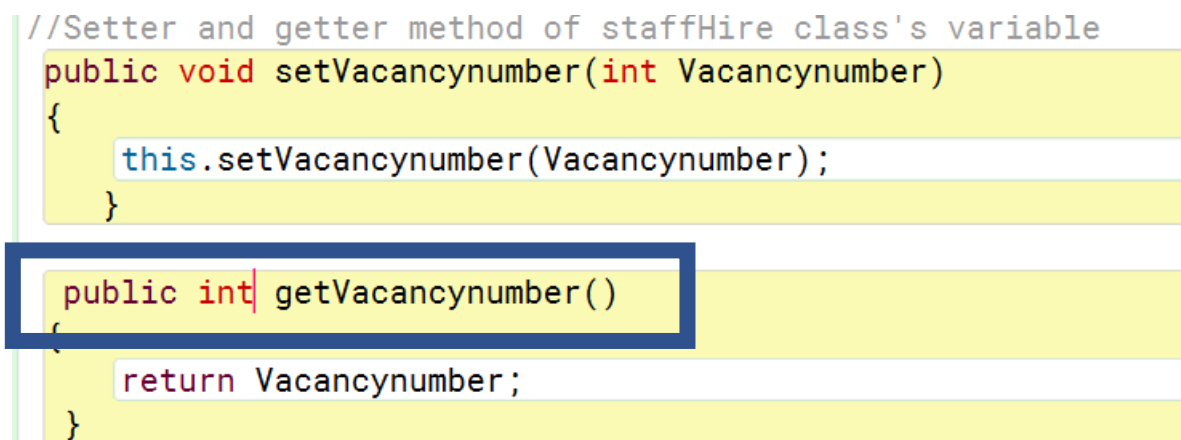


```
//Setter and getter method of staffHire class's variable
public void setVacancynumber(int Vacancynumber)
{
    this.setVacancynumber(Vacancynumber);
}

public String getVacancynumber()
{
    return Vacancynumber;
}

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

Figure 19 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 20 error 2 rectified

While writing code for StaffHire, I encountered issues while declaring the variable String type of variable instead of int variable. To overcome this error I made the return same as previous 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 21 error 3 detection

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

Figure 22 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 complete 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 declaration

    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());//calling 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));
    }
}
```