



Module Code & Module Title
CS4001NI PROGRAMMING

COURSEWORK-1
Staff Hiring System

Assessment Weightage & Type
30% Individual

Year & Semester
2019/20 Autumn

Student Name: Kamal Poudel

Group: L1C1

London Met ID:

College ID:NP01CP4A190102

Assignment Due Date: 13th January 2020

Assignment Submission Date: 13th 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 the late submissions will be treated as non-submission and a mark of zero will be awarded.

Table of Contents:

1	Introduction	1
2	Class Diagram	3
2.1	staffHire Class	5
2.2	partTimeStaffHire Class	6
2.3	fullTimeStaffHire Class	7
3	Pseudocode	8
3.1	Pseudocode of Parent Class: staffHire	8
3.2	Pseudocode of Sub-Class: partTimeStaffHire	10
3.3	Pseudocode of Sub-Class: fullTimeStaffHire	15
4	Method Description	20
4.1	staffHire Class	20
4.2	partTimeStaffHire Class	21
4.3	fullTimeStaffHire Class:	23
5	Tests	25
5.1	Test 1:	25
5.2	Test 2	29
5.3	Test 3	33
5.4	Test 4	36
6	Error detection	41
6.1	Error 1	41
6.2	Error 2	42
6.3	Error 3	43
7	Conclusion	46
8	References	47

9	Appendix.....	48
---	---------------	----

Table of Figures

Figure 1: Java logo.....	1
Figure 2: BlueJ logo	2
Figure 3: staffHire parent class related to partTimeStaffHire and fullTimeStaffHire child classes	2
Figure 4: Inspect Full-Time Staff Class after object creation	26
Figure 5: Introducing variable to the Full-Time Staff Class.....	27
Figure 6: Showing the Staff Name hired for full-time job	27
Figure 7: Reinspection of Full-Time Staff Class	28
Figure 8: Inspection of part-time staff hire after object creation.....	30
Figure 9: Introducing variable to the Part-Time Staff Class	31
Figure 10: Showing the Staff Name hired for part-time job.....	31
Figure 11: Reinspection of Part-Time Staff Class	32
Figure 12: Inspection before termination of part-time staff	34
Figure 13: Terminal output after termination	35
Figure 14: Inspection after Termination.....	35
Figure 15: Inspection of data in full-time staff hire	37
Figure 16: Displaying information of full-time staff hire.....	38
Figure 17: Inspection of information of part-time staff hire	39
Figure 18: Displaying data of part-time staff hire.....	40
Figure 19: Syntax Error with scrambled parameters	41
Figure 20: Supplying parameters in the correct order	42
Figure 21: Data type for job_Type when initializing.....	42
Figure 22: Data type when declaring a method to get job_Type	42
Figure 23: Error handling for wrong data type	43
Figure 24: Run-time error in fullTimeStaff hire	43
Figure 25: Showing the full-time staff hire in BlueJ terminal.....	44
Figure 26: Inspecting the run-time error	44
Figure 27: Solving the run-time error through BlueJ.....	45
Figure 28: Inspecting after the error is solved	45

Table of Tables

Table 1: Class Diagram of parent and child classes of Staff hiring System.....	4
Table 2: Class Diagram of staffHire(Parent) Class.....	5
Table 3: Class Diagram of partTimeStaffHire(Child) Class	6
Table 4: Class Diagram of fullTimeStaffHire(Child) Class	7
Table 5: Inspection of Full-Time Staff Hire	25
Table 6: Inspection of part-time staff hire	29
Table 7: Inspection method for termination	33
Table 8: Showing display method	36

1 Introduction

This is the first coursework of programming which allocates 30% of the total marks of the first semester of first year and is all about the staff hiring system in two different subclasses. i.e. Part-time staff hire and Full-time staff hire. This program is done for the recruitment of an employee in certain organization in accordance to their interest and need. This program is used in order to keep the records of the staffs in managed way. Java is a high-level and object-oriented programming language which is used to define objects and methods assigned to individual class. In java, variables and functions must be clearly defined. The crucial benefit of java is that it limits other sorts of bugs that may be generated by undefined variables or unassigned types. **Source:** (Christensson, 2012)

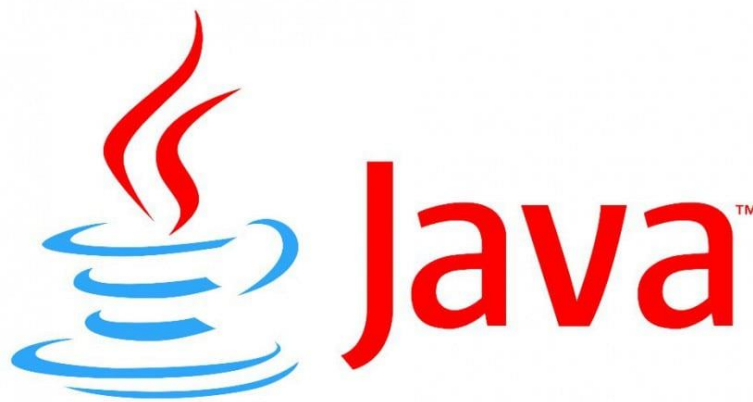


Figure 1: Java logo

This program was done by the help of “BlueJ” app which is a free Java Development designed for beginners. BlueJ helps to interact with objects in the code directly which makes easy to learn concepts or to try something out immediately. **Source:** (Learneroo, 2020) Through this application, the project of staff hiring system known as parent class was made with its child classes named Part-time staff hire and Full-time staff hire.



Figure 2: BlueJ logo

Before writing this program, lots of researches were done to handle the errors and bugs prevailing the program. The main purpose is to develop a Staff hiring program used by different organization for keeping records of an employee such as Vacancy Nmb, Designation, Job type, Salary, joined, qualification and soon. Finally, this program consists of three classes: `staffHire`, `partTimeStaffHire` and `fullTimeStaffHire` where `staffHire` is a parent class and `partTimeStaffHire` and `fullTimeStaffHire` are child classes.

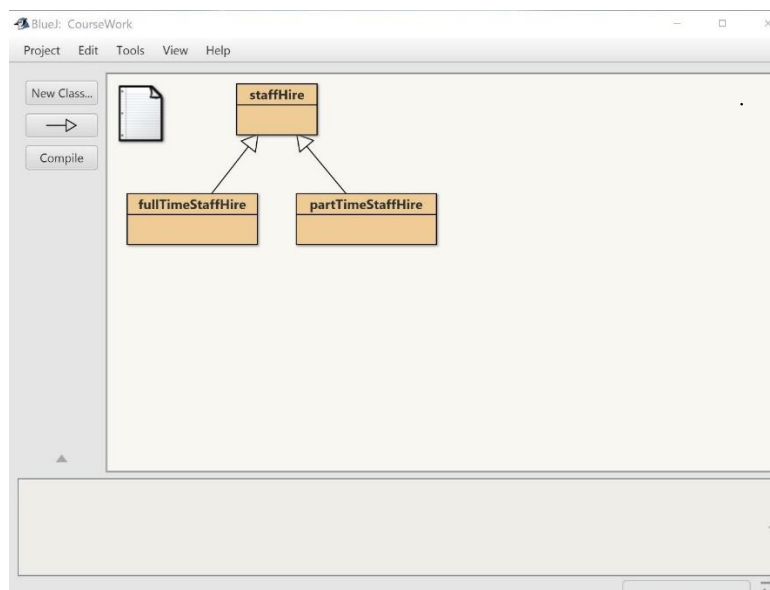


Figure 3: `staffHire` parent class related to `partTimeStaffHire` and `fullTimeStaffHire` child classes

2 Class Diagram

A class diagram is a representation of the connections and source code conditions among classes in the Unified Modeling Language (UML). In this unique situation, a class characterizes the strategies and variables in an object, which is a particular entity in a program or the unit of code speaking to that element. Class diagrams are helpful in all types of object-oriented programming (OOP).

In class diagram, the classes are arranged in groups which have same characteristics and resembles in flowchart manner having three rectangular boxes where top rectangular box contain class name, middle box contains attributes of the class and last box content the methods also known as operations of the class.

Source: (Rouse, 2020)

Here, there are three classes in my project and the class diagram for these are given below:

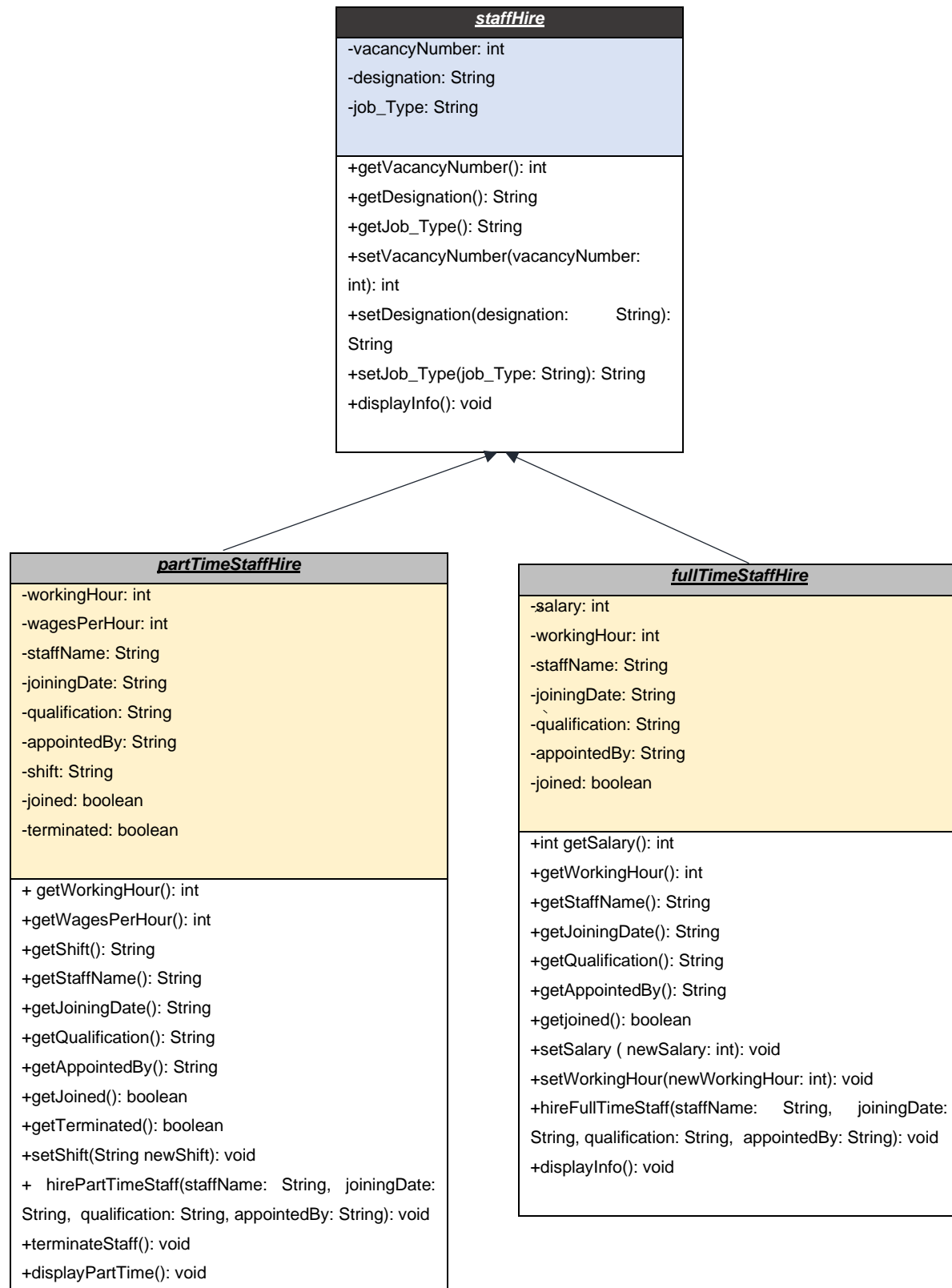


Table 1: Class Diagram of parent and child classes of Staff hiring System

2.1 staffHire Class

This is the parent class of the program and the class diagram of staffHire is shown below:

<u>staffHire</u>
-vacancyNumber: int -designation: String -job_Type: String
+getVacancyNumber(): int +getDesignation(): String +getJob_Type(): String +setVacancyNumber(vacancyNumber: int): int +setDesignation(designation: String): String +setJob_Type(job_Type: String): String +displayInfo(): void

Table 2: Class Diagram of staffHire(Parent) Class

2.2 partTimeStaffHire Class

It is a child class of staffHire and its class diagram is as listed below:

<u>partTimeStaffHire</u>
-workingHour: int -wagesPerHour: int -staffName: String -joiningDate: String -qualification: String -appointedBy: String -shift: String -joined: boolean -terminated: boolean
+ getWorkingHour(): int + getWagesPerHour(): int + getShift(): String + getStaffName(): String + getJoiningDate(): String + getQualification(): String + getAppointedBy(): String + getJoined(): boolean + getTerminated(): boolean + setShift(String newShift): void + hirePartTimeStaff(staffName: String, joiningDate: String, qualification: String, appointedBy: String): void + terminateStaff(): void + displayPartTime(): void

Table 3: Class Diagram of partTimeStaffHire(Child) Class

2.3 fullTimeStaffHire Class

Full-time staff hire class is also a child class of staffHire and its class, attributes and methods are shown in rectangular box as shown in below table:

<u>fullTimeStaffHire</u>
-salary: int -workingHour: int -staffName: String -joiningDate: String -qualification: String -appointedBy: String -joined: boolean
+int getSalary(): int +getWorkingHour(): int +getStaffName(): String +getJoiningDate(): String +getQualification(): String +getAppointedBy(): String +getjoined(): boolean +setSalary(newSalary: int): void +setWorkingHour(newWorkingHour: int): void +hireFullTimeStaff(staffName: String, joiningDate: String, qualification: String, appointedBy: String): void +displayInfo(): void

Table 4: Class Diagram of fullTimeStaffHire(Child) Class

3 Pseudocode

Pseudocode is a term which is regularly utilized in programming and algorithm-based fields. It is a system that enables the programmer to illustrate the usage of an algorithm. Basically, we can say that it's raw material of the algorithm. Regularly now and again, algorithm is done with the assistance of pseudo codes as they can be interpreted by programmer regardless of what their programming background or information is. Pseudocode, as the name proposes, is a false code or a portrayal of code which can be comprehended by even a non-expert with some school level programming knowledge.

As, algorithm is an organized and managed logical sequence of the tasks which helps to solve the problems of the programmer and used technical annotation in their making process while pseudo code is simply an implementation of an algorithm in the form of annotation and informative text written in simple English language. It does not use any type of syntax like any of the programming language and this it can't be compiled or interpreted by the computer. **Source:** (GeeksforGeeks, 2020)

3.1 Pseudocode of Parent Class: staffHire

DEFINE staffHire

DEFINE three instance variables as vacancyNumber, designation, and job_Type.

Create a constructor and initialize the variables.

INITIALIZE vacancyNumber as int type

INITIALIZE designation as String type

INITIALIZE job_Type as String type

DEFINE method getVacancyNumber () as int type

DO

Extract the vacancyNumber

Return vacancyNumber

END DO

DEFINE method getDesignation () as String type

DO

Extract designation

Return designation

END DO

DEFINE method getJob_Type () as String Type

DO

Extract job_Type

Return job_Type

END DO

DEFINE method setVacancyNumber (accept newvalue as argument of int Type)

DO

Set the value of vacancyNumber

END DO

DEFINE method setDesignation (accept newname as argument of String Type)

DO

Set the value of designation

END DO

DEFINE method setJob_Type (accept newvalue as argument of String Type)

DO

Set the value of job_Type

END DO

DEFINE method displayInfo ()

DO

Display (.....)

Display (Vacancy Number)

Display (Designation)

Display (Job Type)

Display (.....)

END DO

3.2 Pseudocode of Sub-Class: partTimeStaffHire

DEFINE partTimeStaffHire

DEFINE instance variables as workingHour, wagesPerHour, staffName, joiningDate, qualification, appointedby, shift, joined, and terminated.

Create a constructor and initialize the variables.

INITIALIZE workingHour as int type

INITIALIZE wagesPerHour as int type

INITIALIZE staffName as String type

INITIALIZE joiningDate as String type

INITIALIZE qualification as String type

INITIALIZE appointedBy as String type

INITIALIZE joined as boolean type

INITIALIZE terminated as boolean type

DEFINE method getWorkingHour () as int type

DO

Extract the workingHour

Return workingHour

END DO

DEFINE method getWagesPerHour () as int type

DO

Extract wagesPerHour

Return wagesPerHour

END DO

DEFINE method getShift () as String Type

DO

Extract shift

Return shift

END DO

DEFINE method getStaffName () as String type

DO

Extract staffName

Return staffName

END DO

DEFINE method getJoiningDate () as int type

DO

Extract joiningDate

Return joiningDate

END DO

DEFINE method getQualification () as String type

DO

Extract qualification

Return qualification

END DO

DEFINE method getAppointedBy () as String type

DO

Extract appointedBy

Return appointedBy

END DO

DEFINE method getJoined () as boolean type

DO

Extract joined

Return joined

END DO

DEFINE method getTerminated () as boolean type

DO

Extract terminated

Return terminated

END DO

DEFINE method setShift (accept newShift as argument of String Type)

DO

IF (joined is false)

Display (newShift)

END IF

ELSE

Display (Staff has already been hired)

END ELSE

END DO

DEFINE method hirePartTimeStaff (accept staffName, joiningDate, qualification, appointedBy as argument of String Type)

DO

IF (joined is true)

Display (Staff already been hired)

Set value terminated is true

END IF

ELSE

INITIALIZE value to staffName

INITIALIZE value to joiningDate

INITIALIZE value to qualification

INITIALIZE value to appointedBy

INITIALIZE joined is true

INITIALIZE terminated is false

Display (staffName as hired for part time job)

```
        END ELSE

    END DO

    DEFINE method terminateStaff ()

    DO

        IF (terminated is true)

            Display (Staff records can't find)

        END IF

        ELSE

            Display (staff has been removed from the job)

            INITIALIZE staffName to empty String

            INITIALIZE joiningDate to empty String

            INITIALIZE qualification to empty String

            INITIALIZE appointedBy to empty String

            INITIALIZE joined to false

            INITIALIZE      terminated      to      true

        END ELSE

    END DO

    DEFINE method displayPartTime()

    DO

        Call displayInfo() from parent class

        IF

            Display (.....)
```

Display (Vacancy Number)

`Display (Designation)

Display (Job Type)

Display (Staff Name)

Display (Wages per Hour)

Display (Working Hour)

Display (Joined Date)

Display (Qualification)

Display (Appointed By)

Display (Income per Day)

Display (.....)

END IF

ELSE

Display (staff has not hired for part time job)

END ELSE

END DO

3.3 Pseudocode of Sub-Class: fullTimeStaffHire

DEFINE fullTimeStaffHire

DEFINE seven instance variables as salary, workingHour, staffName, joiningDate, qualification, appointedBy, and joined

Create a constructor and initialize the variables

INITIALIZE salary as int type

INITIALIZE workingHour as int type

INITIALIZE staffName as String type

INITIALIZE joiningDate as String type

INITIALIZE qualification as String type

INITIALIZE appointedBy as String type

INITIALIZE joined as boolean type

DEFINE method getSalary() as int type

DO

Extract Salary

Return salary

END DO

DEFINE method getWorkingHour() as int type

DO

Extract workingHour

Return workingHour

END DO

DEFINE method getStaffName () as String type

DO

Extract staffName

Return staffName

END DO

DEFINE method getJoiningDate () as String type

DO

Extract joiningDate

Return joiningDate

END DO

DEFINE method getQualification() as String type

DO

Extract qualification

Return qualification

END DO

DEFINE method getAppointedBy() as String Type

DO

Extract appointedBy

Return appointedBy

END DO

DEFINE method getjoined() as boolean type

DO

Extract joined value

Return joined value

END DO

DEFINE method setSalary (newsalary as argument of int type)

DO

IF (joined is false)

Set value newsalary to salary

Display (New Salary)

END IF

ELSE

Display (Impossible to change salary as staff already hired)

END ELSE

END DO

DEFINE method setWorkingHour (newWorkingHour as argument of int type)

DO

IF (joined is false)

Set value newWorkingHour to workingHour

Display (New Working hour)

END IF

ELSE

Display (Impossible to change salary as staff already hired)

END ELSE

END DO

DEFINE method hireFullTimeStaff (StaffName, JoiningDate, Qualification, AppointedBy as arguments of String Type)

DO

INITIALIZE staffName value to StaffName

INITIALIZE joiningDate value to JoiningDate

INITIALIZE qualification value to Qualification

```
INITIALIZE appointedBy value to appointedBy

    IF (joined is false)
        Display (StaffName that is appointedBy)
    END IF

    ELSE
        Display (Staff has already hired)
    END ELSE

END DO

DEFINE method displayInfo()

DO

    Call displayInfo() {calling super class display method}

    IF (joined is true)
        Display (.....)
        Display (Staff Name)
        Display (Salary of staff)
        Display (Working hour of staff)
        Display (Joined Date of staff)
        Display (qualification of staff)
        Display (staff is appointedBy)
        Display (.....)
    END IF

END DO
```


4 Method Description

A method is a collection of statements that perform some specific task and return the result to the caller. **Source:** (GeeksforGeeks, 2020) In this program, different methods are used in three different classes which are listed below:

4.1 staffHire Class

Methods are required to assign, call and update the values and display the result. Different methods are used in this class and are illustrate below with suitable comments:

❖ `getVacancyNumber():`

It is a type of getter method. This method returns the vacancyNumber and its return type is int.

❖ `setVacancyNumber()`

It is a type of setter method and helps to modify the value through given parameter with return type int.

❖ `getDesignation()`

This method gets the value stored in designation variable of the current instance of the class and its return type is String.

❖ `setDesignation()`

A mutator is a method that updates the value of designation as its return type is String.

❖ `getJob_Type()`

An accessor method that reads the value of job type in staffHire class and its return type is String.

❖ `setJob_Type()`

It is used to change the value of the job type through parameter passed through it and its return type is String.

❖ `displayInfo()`

This method displays the details of the staff hire stored on the current instance of the class, which is void return type, and shows the value of vacancyNumber, designation and job_Type, if its value is already set.

4.2 partTimeStaffHire Class

Here, in this class various method are used to run the program and they are listed below:

- ❖ getWorkingHour()

It is an accessor method which is used to return working hour whose return type is int.

- ❖ getWagesPerHour()

It is used to access wages per hour of the object with the return type int.

- ❖ getShift()

It is a type of getter method. This method returns the shift having return type as String.

- ❖ getStaffName()

An accessor method that reads the value of staff name whose return type is String.

- ❖ getJoiningDate()

This method is used to return the value of joining date of staff and its return type is String.

- ❖ getQualification()

It is a type of getter method and helps to retain the value of qualification where its return type is String.

- ❖ getAppointedBy()

An accessor method that reads the value of appointed by of the current instance of the class with return type String.

❖ `getJoined()`

It is used to return joined variable of the instance class where return type is boolean.

❖ `getTerminated()`

This type of accessor method is used to return terminated variable of the instance class having return type is boolean.

❖ `setShift()`

This method takes a String value into the salary variable in local scope and sets the value of the salary variable of the current instance of the class.

❖ `hirePartTimeStaff()`

This method is used to know whether staff has been already hired or not. If the staff hired has already been done or yet to be hired, then it returns a corresponding output to the user. This method uses `staffName`, `joiningDate`, `qualification`, `appointedBy` as String type. Lastly, the boolean values of `joined` and `terminated` are changed accordingly.

❖ `terminateStaff()`

This method is used to terminate the staff. If the staff hire has already been terminated it displays a corresponding information and if not terminates the staff. Lastly, the boolean values of `terminated` and `joined` are changed accordingly.

❖ `displayPartTime()`

This method is used to display all the information on the present instance of the class. If the `joined` value is true, it display the value of `displayInfo()` method of `staffHire` (parent) class and it also calls the `display()` method of the super class as well. This method displays vacancy number, designation, job type of suber class and staff name, wages per hour, joined date, qualification, appointed by, income per day is printed.

4.3 fullTimeStaffHire Class:

Different methods are used in this class and are display below with suitable comments:

❖ `getSalary()`

This type of accessor method is used to return salary having return type int.

❖ `getWorkingHour()`

It is a getter method which is used to return working hour variable of the current instance of the class where return type is int.

❖ `getStaffName()`

An accessor method that reads the value of staff name variable from current instance class, having return type is String.

❖ `getJoiningDate()`

This method is used to return the value of joining date of staff where return type is String.

❖ `getQualification()`

It is a type of getter method and helps to retain the value of qualification where its return type is String.

❖ `getAppointedBy()`

An accessor method that reads the value of appointed by variable of the current instance of class where return type is String.

❖ `getjoined()`

It is a method to get the value stored in the joined variable of the current instance of the class having return type is boolean.

❖ `setSalary ()`

It is a method to get the set the value of salary variable of the current instance of the class where boolean joined is false. Furthermore, if the staff has already joined, it will say that the change is not possible.

❖ `setWorkingHour()`

If boolean `joined` is false, it display the method to set the value of `workingHour` variable of the current instance of the class. Otherwise, it will say that the change is not possible.

❖ `hireFullTimeStaff()`

It is a method to hire the staff for full time. In additional, if the `joined` status is false then the method displays the staff name, appointed by, joining date, qualification as String type and boolean `joined` status will be true. Otherwise, it shows the corresponding information.

❖ `displayInfo()`

This method has the same signature as that of the super class. This method first call to the super class for its display method. If boolean `joined` is true, it prints out the corresponding parameters such as `staffName`, `salary`, `workingHour`, `joiningDate`, `qualification` and `appointedby`. Otherwise, it shows the corresponding result as staff has not hired yet for full-time job.

5 Tests

In this testing field, we are going to test different class according to the question asked to do. Here, we are going to test fullTimeStaffHire class, partTimeStaffHire class, terminating the staff of partTimeStaffHire class and displaying the outcome of fullTimeStaffHire and partTimeStaffHire classes.

5.1 Test 1:

Inspect fullTimeStaffHire Class, appoint full-time staff, and reinspect the fullTimeStaffHire Class:

Objective	To inspect whether the method hireFullStaffHire
Action	<ul style="list-style-type: none"> ✚ Firstly, fullTimeStaffHire is executed and object is created. ✚ Inspect the object. ✚ Calling the hireFullTimeStaff() method. ✚ Re- inspecting it.
Expected Result	All the data supplied for running the function hireFullTimeStaff() is updated after inspection
Actual Result	Every data supplied during the running of the function was updated during inspection
Conclusion	The test was successful as full-time staff has been hired

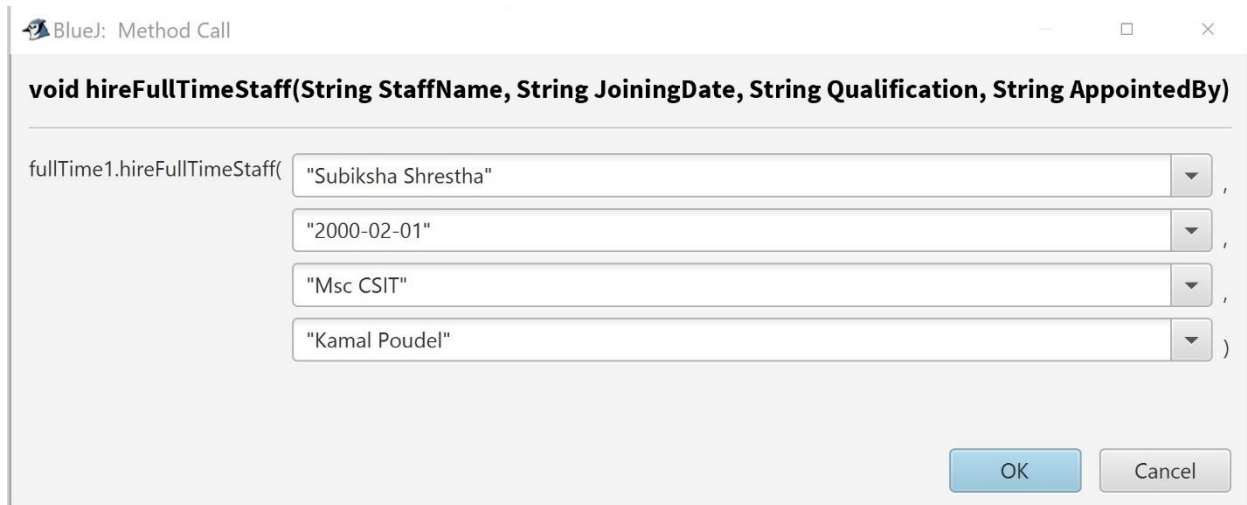
Table 5: Inspection of Full-Time Staff Hire

fullTime1 : fullTimeStaffHire

private int salary	80000	Inspect
private int workingHour	5	
private String staffName	""	Get
private String joiningDate	""	
private String qualification	""	
private String appointedBy	""	
private boolean joined	false	
private int vacancyNumber	1	
private String designation	"Senior Developer"	
private String job_Type	"Full-Time Job"	

Show static fields Close

Figure 4: Inspect Full-Time Staff Class after object creation



BlueJ: Method Call

void hireFullTimeStaff(String StaffName, String JoiningDate, String Qualification, String AppointedBy)

fullTime1.hireFullTimeStaff("Subiksha Shrestha" ,
"2000-02-01"
"Msc CSIT"
"Kamal Poudel")

OK Cancel

Figure 5: Introducing variable to the Full-Time Staff Class



BlueJ: Terminal Window - CourseWork

Options

The staff Subiksha Shrestha is hired by Kamal Poudel for full-time job.

Can only enter input while your programming is running

Figure 6: Showing the Staff Name hired for full-time job



Figure 7: Reinspection of Full-Time Staff Class

5.2 Test 2

Inspect partTimeStaffHire Class, appoint part-time staff, and reinspect the partTimeStaffHire Class:

Objective	To inspect whether the method hirePartStaffHire
Action	<ul style="list-style-type: none"> ✚ Firstly, partTimeStaffHire is executed and object is created. ✚ Inspect the object. ✚ Calling the hirepartTimeStaff() method. ✚ Re- inspecting it.
Expected Result	All the data supplied for running the function hirePartTimeStaff() is updated after inspection
Actual Result	Every data supplied during the running of the function was updated during inspection
Conclusion	The test was successful as part-time staff has been hired

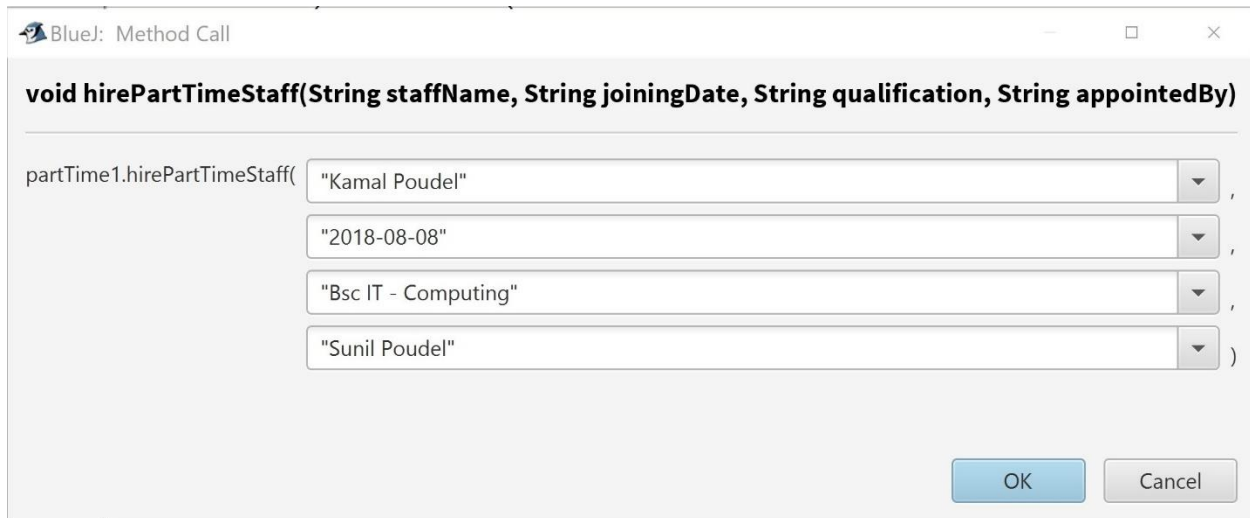
Table 6: Inspection of part-time staff hire

partTime1 : partTimeStaffHire

private int workingHour	3	Inspect
private int wagesPerHour	12000	
private String staffName	""	Get
private String joiningDate	""	
private String qualification	""	
private String appointedBy	""	
private String shift	"Night"	
private boolean joined	false	
private boolean terminated	false	
private int vacancyNumber	1	
private String designation	"Junior Developer"	
private String job_Type	"Part-Time Job"	

Show static fields Close

Figure 8: Inspection of part-time staff hire after object creation



BlueJ: Method Call

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

partTime1.hirePartTimeStaff("Kamal Poudel" ,
"2018-08-08" ,
"Bsc IT - Computing" ,
"Sunil Poudel")

OK Cancel

Figure 9: Introducing variable to the Part-Time Staff Class



BlueJ: Terminal Window - CourseWork

Options

The Staff Kamal Poudel has been hired for a part-time job.

Can only enter input while your programming is running

Figure 10: Showing the Staff Name hired for part-time job

partTime1 : partTimeStaffHire

private int workingHour	3	Inspect
private int wagesPerHour	12000	
private String staffName	"Kamal Poudel"	Get
private String joiningDate	"2018-08-08"	
private String qualification	"Bsc IT - Computing"	
private String appointedBy	"Sunil Poudel"	
private String shift	"Night"	
private boolean joined	true	
private boolean terminated	false	
private int vacancyNumber	1	
private String designation	"Junior Developer"	
private String job_Type	"Part-Time Job"	

Show static fields Close

Figure 11: Reinspection of Part-Time Staff Class

5.3 Test 3

Inspect partTimeStaffHire Class, change the termination status of a staff, and reinspect the partTimeStaffHire Class.

Objective	Checking status of termination status after terminating in partTimestaffHire class
Action	<ul style="list-style-type: none"> ✚ Firstly, partTimeStaffHire is executed and object is created. ✚ Inspect the object. ✚ Calling the terminateStaff() method. ✚ Re- inspecting it.
Expected Result	The terminated variable of the object needs to be changed and return boolean must be true
Actual Result	The value of terminated variable changes to true
Conclusion	The test was successful as termination is changed from false to true

Table 7: Inspection method for termination

partTime1 : partTimeStaffHire

private int workingHour	3	Inspect
private int wagesPerHour	12000	
private String staffName	"Kamal Poudel"	Get
private String joiningDate	"2018-08-08"	
private String qualification	"Bsc IT - Computing"	
private String appointedBy	"Sunil Poudel"	
private String shift	"Night"	
private boolean joined	true	
private boolean terminated	false	
private int vacancyNumber	1	
private String designation	"Junior Developer"	
private String job_Type	"Part-Time Job"	

Show static fields Close

Figure 12: Inspection before termination of part-time staff

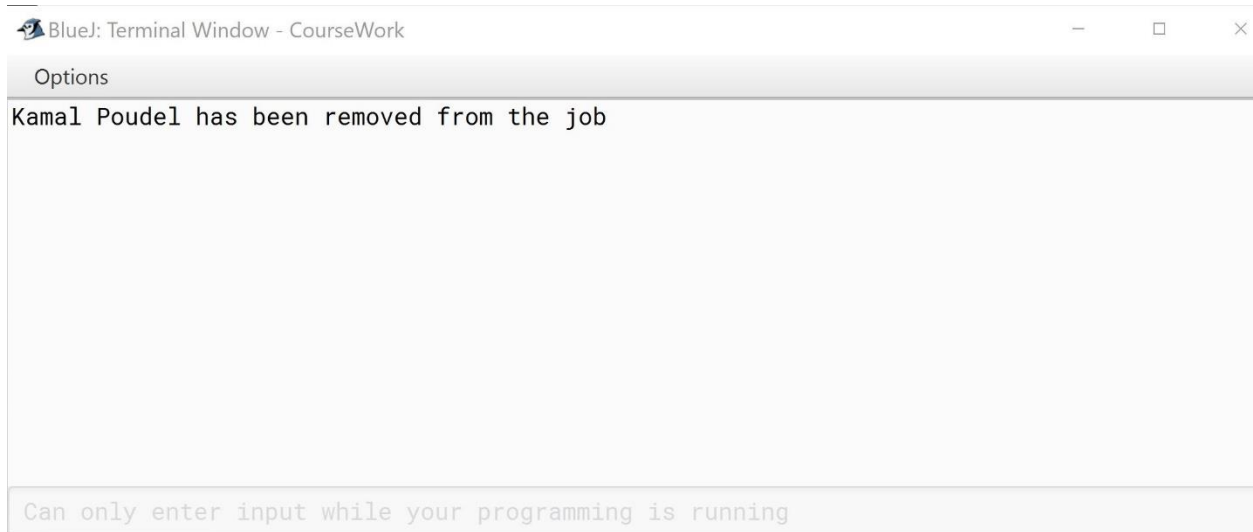


Figure 13: Terminal output after termination

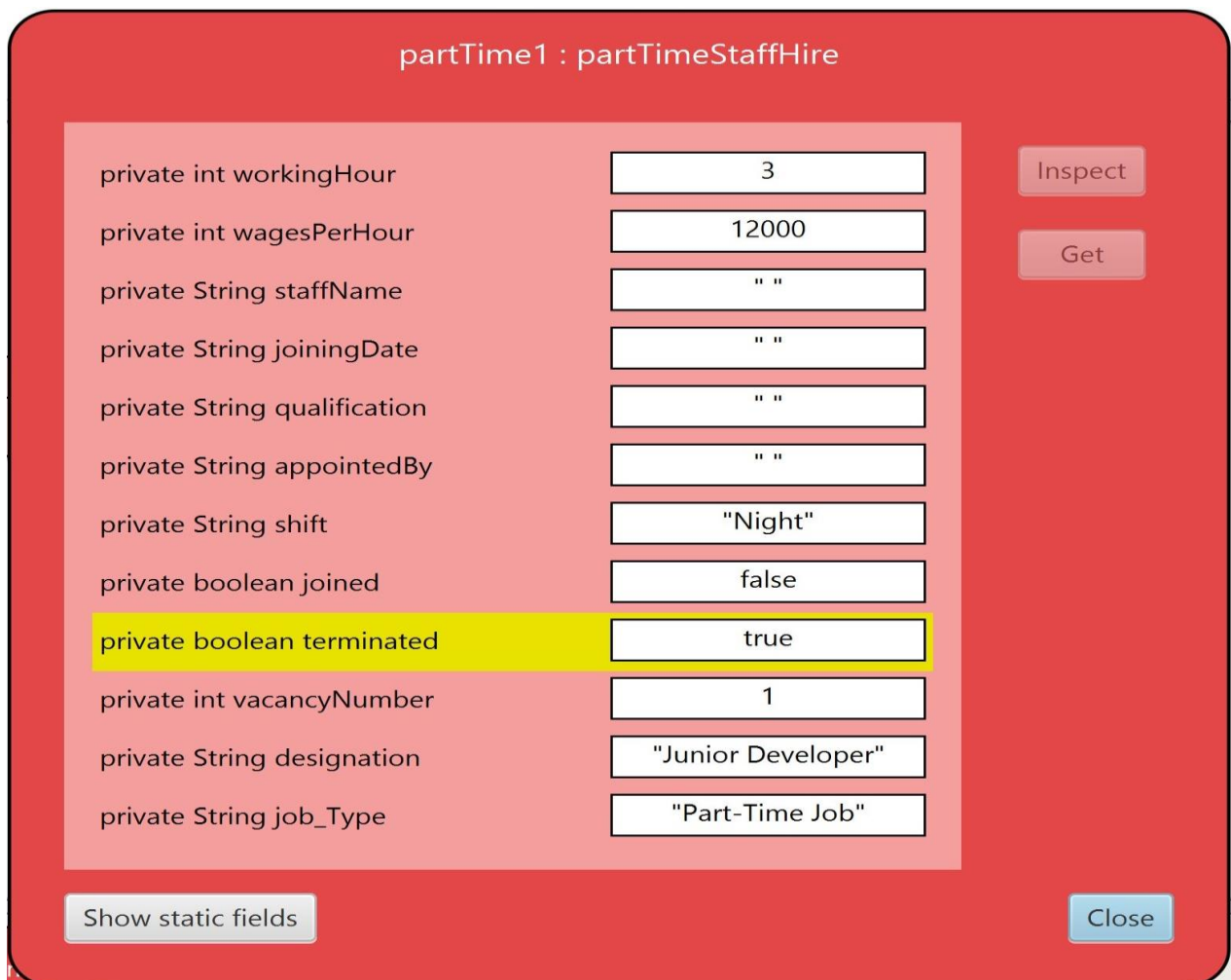


Figure 14: Inspection after Termination

5.4 Test 4

Displays the details of partTimeStaffHire and fullTimeStaffHire Classes.

Objective	To display information of staff on full-time staff hire and part-time staff hire class
Action	<ul style="list-style-type: none"> Firstly,fullTimeStaffHire and partTimeStaffHire is executed and object is created. Calling the hireFullTimeStaffHire() and hirePartTimeStaffHire() method. Lastly, the method displayInfo() of both classes are executed
Expected Result	All the stored data of partTimeStaffHire and fullTimeStaffHire should be printed properly
Actual Result	Both the output of partTimeStaffHire and fullTimeStaffHire were displayed in BlueJ terminal
Conclusion	The test was successfully done

Table 8: Showing display method

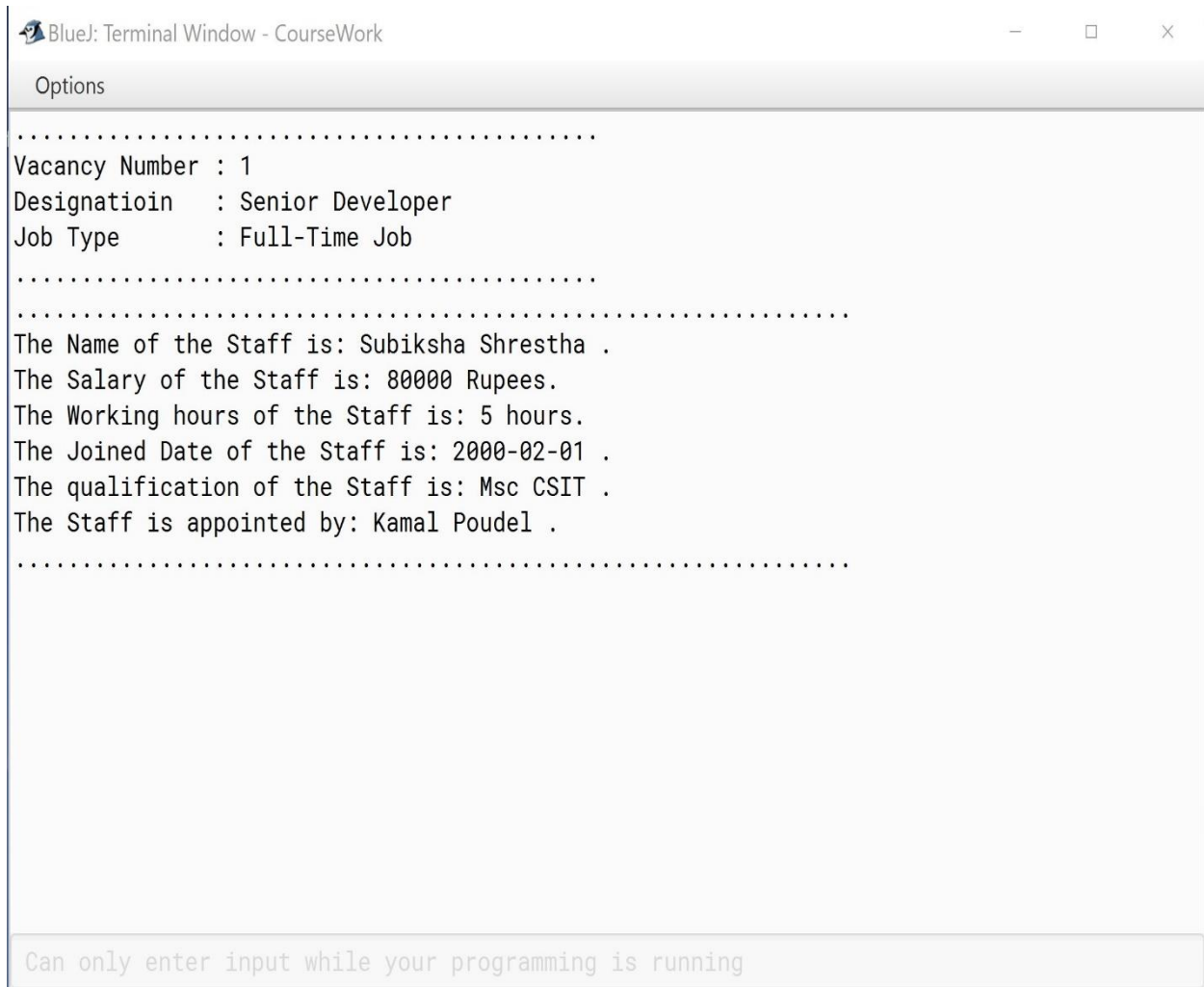
fullTime1 : fullTimeStaffHire

private int salary	80000	Inspect
private int workingHour	5	
private String staffName	"Subiksha Shrestha"	Get
private String joiningDate	"2000-02-01"	
private String qualification	"Msc CSIT"	
private String appointedBy	"Kamal Poudel"	
private boolean joined	true	
private int vacancyNumber	1	
private String designation	"Senior Developer"	
private String job_Type	"Full-Time Job"	

Show static fields

Close

Figure 15: Inspection of data in full-time staff hire



The screenshot shows a BlueJ Terminal Window titled "BlueJ: Terminal Window - CourseWork". The window has a title bar with standard minimize, maximize, and close buttons. Below the title bar is a tab labeled "Options". The main area of the terminal displays the following text:

```
.....  
Vacancy Number : 1  
Designatioin   : Senior Developer  
Job Type       : Full-Time Job  
.....  
.....  
The Name of the Staff is: Subiksha Shrestha .  
The Salary of the Staff is: 80000 Rupees.  
The Working hours of the Staff is: 5 hours.  
The Joined Date of the Staff is: 2000-02-01 .  
The qualification of the Staff is: Msc CSIT .  
The Staff is appointed by: Kamal Poudel .  
.....  
  
Can only enter input while your programming is running
```

Figure 16: Displaying information of full-time staff hire

partTime1 : partTimeStaffHire

private int workingHour	3	Inspect
private int wagesPerHour	12000	
private String staffName	"Kamal Poudel"	Get
private String joiningDate	"2018-08-08"	
private String qualification	"Bsc IT - Computing"	
private String appointedBy	"Sunil Poudel"	
private String shift	"Night"	
private boolean joined	true	
private boolean terminated	false	
private int vacancyNumber	1	
private String designation	"Junior Developer"	
private String job_Type	"Part-Time Job"	

Show static fields Close

Figure 17: Inspection of information of part-time staff hire



BlueJ: Terminal Window - CourseWork

Options

```
.....  
Vacancy Number : 1  
Designatioin   : Junior Developer  
Job Type       : Part-Time Job  
.....  
.....  
Vacancy Number: 1  
Designation: Junior Developer  
Job Type: Part-Time Job  
Staff Name: Kamal Poudel  
Wages per Hour: 12000  
Working Hour: 3  
Joined Date: 2018-08-08  
Qualification: Bsc IT - Computing  
Appointed By: Sunil Poudel  
Income per Day: 36000  
.....
```

Can only enter input while your programming is running

Figure 18: Displaying data of part-time staff hire

6 Error detection

Anybody engaged with computer programming, even (maybe particularly) beginners are going to experience blunders and bugs of different kinds that force them to chase down the guilty party bit of code and make the important changes. Here, are three main computer coding error, they are logical/run-time errors, syntax errors and semantic errors. Shortly, they are described below;

✓ **Logical/run-time error:**

This is the hardest error to detect as this error is only found after running the program. This type of error can only detect when the value is given but the output will be either null or boolean value is altered. This type of error is shown below in **Error 3**.

✓ **Syntax Error:**

This type of error is found at compile time, and it must be corrected before the program can run. Such error is given below at **Error 1**.

✓ **Semantic Error:**

This error is known as improper uses of “program statements” and produces meaningless data at all while running the program. **Source:** (htkadm, 2017)

6.1 Error 1

One of the errors that I encountered was the scrambled order of parameters (known as s error) while calling the constructor of the super class. During compiling, it shows a bug.

```
public partTimeStaffHire(int vacancyNumber, String designation, String job_Type, int workingHour, int wagesPerHour, String Shift)
{
    super( designation,vacancyNumber, job_Type);// invoking the super class by using super keyword
    this.workingHour=workingHour;//this keyword is used to initiate current class constructor
    this.wagesPerHour=wagesPerHour;
    this.shift=Shift;
}
```

Figure 19: Syntax Error with scrambled parameters

The error was very easy to debug as it only required the order of parameters to be changed. As, designation have String data type and vacancyNumber has int data type. Thus, to solve this bug, only the variables were properly managed in their data type.

```

public partTimeStaffHire(int vacancyNumber, String designation, String job_Type, int workingHour, int wagesPerHour, String Shift)
{
    super(vacancyNumber, designation, job_Type); // invoking the super class by using super keyword
    this.workingHour=workingHour; // this keyword is used to initiate current class constructor
    this.wagesPerHour=wagesPerHour;
    this.shift=Shift;
}

```

Figure 20: Supplying parameters in the correct order

6.2 Error 2

Another type of error I encountered, was when data types of variables like job_Type is String but mixed with int type.

```

public class staffHire
{
    // private is used only for visible within the class, not from any c
    private int vacancyNumber;
    private String designation;
    private String job_Type;
}

```

Figure 21: Data type for job_Type when initializing

In this declaring method, the job_type have String data type but when it is initialized in get method, the data type of job_Type has been written as int. When it is compiled, error is found as shown in below figure.

```

public int getJob_Type()
{
    return job_Type;
}

```

Figure 22: Data type when declaring a method to get job_Type

After knowing the bug, the data type of job_Type has been changed from int to String data type and the detected error is solved and its screenshot is given below.

```

public String getJob_Type()
{
    return job_Type;
}

```

Figure 23: Error handling for wrong data type

6.3 Error 3

The most confusing part of this error was encountered as run-time error. This error was detected after inspecting the fullTimeStaffHire method.

```

public void hireFullTimeStaff(String StaffName, String JoiningDate, String Qualification, String AppointedBy)
{
    this.staffName=StaffName;
    this.joiningDate=JoiningDate;
    this.qualification=Qualification;
    this.appointedBy=AppointedBy;
    if(joined==false)
    {
        System.out.println("The staff " + StaffName + " is hired by " + AppointedBy + " for full-time job." );
    }else{
        System.out.println("Sorry!! The staff has already been hired.");
    }
}

//Showing the information that we entered for full time staff hire!!!
public void displayInfo()
{
    super.displayInfo();
    if(joined==true)
    {
        System.out.println(".....");
        System.out.println("The Name of the Staff is: " + getStaffName()+" .");
        System.out.println("The Salary of the Staff is: " + getSalary()+" Rupees.");
        System.out.println("The Working hours of the Staff is: " + getWorkingHour()+" hours.");
        System.out.println("The Joined Date of the Staff is: " + getJoiningDate()+" .");
        System.out.println("The qualification of the Staff is: " + getQualification()+" .");
        System.out.println("The Staff is appointed by: " + appointedBy+" .");
        System.out.println(".....");
    }
}

```

omplied - no syntax errors

Figure 24: Run-time error in fullTimeStaff hire

Here, after calling the method hireFullTimeStaff(), staff name and appointed by has been seen in BlueJ terminal as shown below.

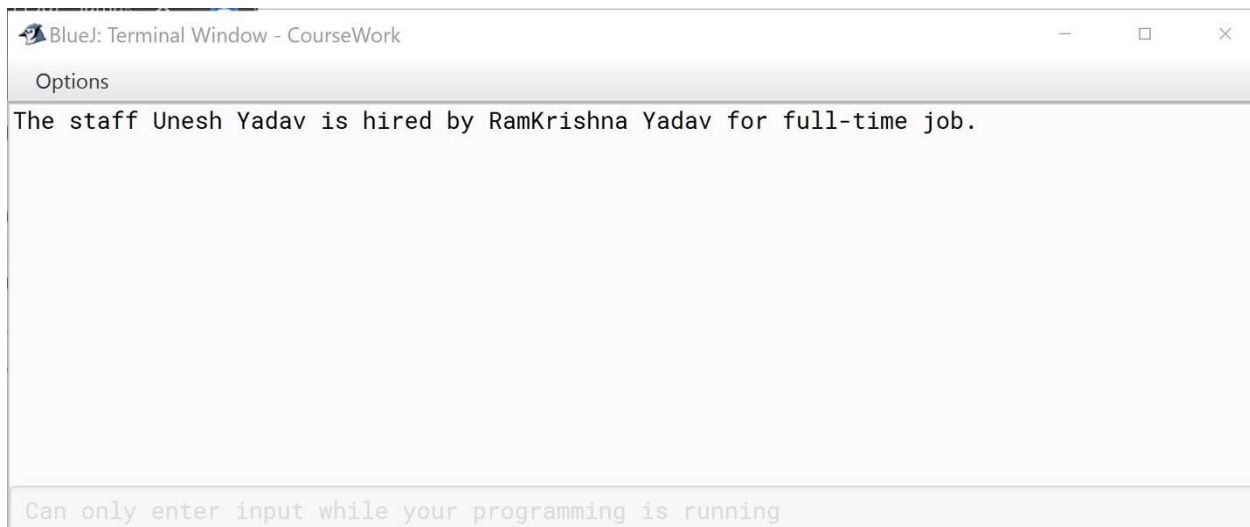


Figure 25: Showing the full-time staff hire in BlueJ terminal

While inspecting the fullTimeStaff, all the value that are set are filled in their respective fields but after hiring the staff, the boolean joined is false which is a run-time error.

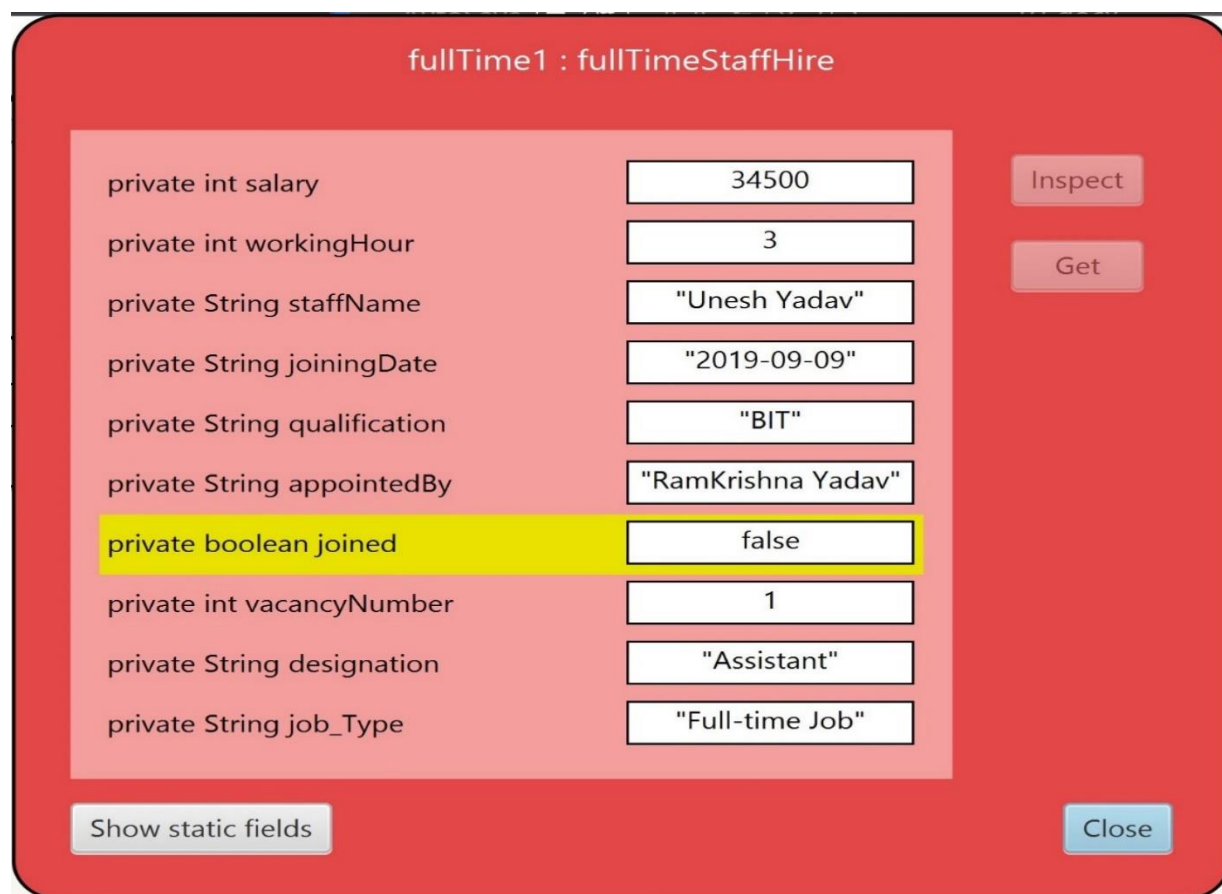


Figure 26: Inspecting the run-time error

After detecting the error, in this method “this.joined=true” has been inserted in order to overcome the bug and after hiring the full-time staff the boolean joined will be changed to true from false.

```
public void hireFullTimeStaff(String StaffName, String JoiningDate, String Qualification, String AppointedBy)
{
    if(joined==false)
    {
        System.out.println("The staff " + StaffName + " is hired by " + AppointedBy + " for full-time job.");
        this.staffName=StaffName;
        this.joiningDate=JoiningDate;
        this.qualification=Qualification;
        this.appointedBy=AppointedBy;
        this.joined=true;
    }else{
        System.out.println("Sorry!! The staff has already been hired.");
    }
}
```

Figure 27: Solving the run-time error through BlueJ

After solving the bug, the staff has been hired and inspect it which shows all the information of the staff and the Boolean joined has been changed as true and the error has been solved.

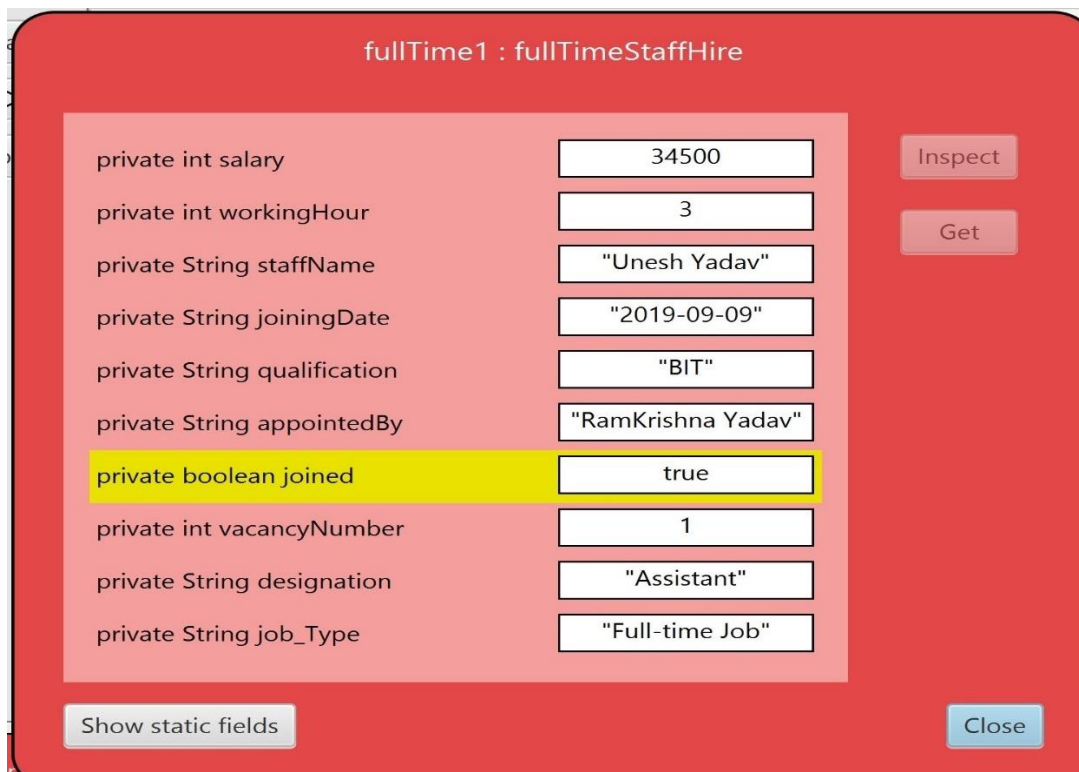


Figure 28: Inspecting after the error is solved

7 Conclusion

This coursework was concluded by keen research, teacher consultation and group discussion. The very coursework was given in order to inspect our knowledge about java programming. This coursework has been able to enhance our knowledge about creating objects and running methods. This coursework was about developing a project in order to know the present situation of the staff. This project helps to find out whether the staffs are hired or not, different attributes and running different tests. This coursework made me able to explore a lot about programming. While doing this coursework, many errors were found but also were successfully overcome.

For this coursework, various research has been done. Different websites, article, journal, books, tutorial videos were referenced. In this project, staffHire is the super or parent class whereas fullTimeStaffHire and partTimeStaffHire are its sub-classes or child classes. Different codes were written in order to match the requirements. Finally, the coursework was completed which helped to flourish my knowledge about java programming.

8 References

Christensson, P., 2012. *Java Definition*. [Online] Available at: <https://techterms.com/definition/java> [Accessed 05 01 2020].

GeeksforGeeks, 2020. *How to write a Pseudo Code - GeeksforGeeks*. [Online] Available at: <https://www.geeksforgeeks.org/how-to-write-a-pseudo-code/> [Accessed 01 01 2020].

GeeksforGeeks, 2020. *Methods in Java - GeeksforGeeks*. [Online] Available at: <https://www.geeksforgeeks.org/methods-in-java/> [Accessed 10 Jan 2020].

htkadm, 2017. *The 3 Basic Types of Programming Errors - Hedge Think*. [Online] Available at: <https://www.hedgethink.com/3-basic-types-programming-errors/> [Accessed 13 01 2019].

Learneroo, 2020. *Classes and Objects in BlueJ - Learneroo*. [Online] Available at: <https://www.learneroo.com/modules/18/nodes/122> [Accessed 05 January 2020].

Rouse, M., 2020. *What is a class diagram_ Definition from WhatIs.com*. [Online] Available at: <https://searchapparchitecture.techtarget.com/definition/class-diagram> [Accessed 04 01 2020].

9 Appendix

- staffHire

```
public class staffHire
{
    // private is used only for visible within the class, not from any other class(including
    subclasses)

    private int vacancyNumber;

    private String designation;

    private String job_Type;

    //creation of constructor which takes parameters of integer, str and str

    public staffHire(int vacancyNumber,String designation, String job_Type)
    {
        //this keyword is used to invoke current class constructor

        this.vacancyNumber=vacancyNumber;

        this.designation=designation;

        this.job_Type=job_Type;

    }

    // getter and setter method for the variables to use them from the next classees as private
    cannot be accessed in other class unless we use set and get method

    //get helps to return the value!!!!

    public int getVacancyNumber()
```

```
{  
    return vacancyNumber;  
}  
  
//set helps to modify the value  
public void setVacancyNumber(int vacancyNumber)  
{  
    this.vacancyNumber=vacancyNumber;  
}  
  
//get helps to return the value!!!!  
public String getDesignation()  
{  
    return designation;  
}  
  
//set helps to modify the value  
public void setDesignation(String designation)  
{  
    this.designation=designation;  
}  
  
//get helps to return the value!!!!  
public String getJob_Type()  
{  
    return job_Type;  
}
```

```
//set helps to modify the value

public void setJob_Type(String job_Type)

{

    this.job_Type=job_Type;

}

//displaying the inormation

public void displayInfo()

{

    System.out.println(".....");

    System.out.println("Vacancy Number : " +getVacancyNumber());

    System.out.println("Designatioin  : " +getDesignation());

    System.out.println("Job Type      : " +getJob_Type() );

    System.out.println(".....");

}

}
```

- partTimeStaffHire

//extends keyword is used to indicate that a new class is derived from the base class using inheritance

```
public class partTimeStaffHire extends staffHire

{
```

// private is used only for visible within the class, not from any other class(including subclasses)

```
private int workingHour;
```

```
private int wagesPerHour;
```

```
private String staffName;
```

```
private String joiningDate;
```

```
private String qualification;
```

```
private String appointedBy;
```

```
private String shift;
```

```
private boolean joined;//boolean is used in conditional statements
```

```
private boolean terminated;
```

//calling constructor as constructor donot use void because constructor returns the object it creates not void.

```
public partTimeStaffHire(int vacancyNumber, String designation, String job_Type, int workingHour, int wagesPerHour, String Shift)
```

```
{
```

```
    super(vacancyNumber, designation, job_Type);// invoking the super class by using super keyword
```

```
    this.workingHour=workingHour;//this keyword is used to initiate current class constructor
```

```
    this.wagesPerHour=wagesPerHour;
```

```
    this.shift=Shift;
```

```
    staffName="";
```

```
    joiningDate="";
```



```
        qualification="";
        appointedBy="";
        joined=false;
        terminated=false;
    }

    //getter is a method that reads value of a variable
    public int getWorkingHour()
    {
        return workingHour;
    }

    public int getWagesPerHour()
    {
        return wagesPerHour;
    }

    public String getShift()
    {
        return shift;
    }

    public String getStaffName()
    {
        return staffName;
    }

    public String getJoiningDate()
```

```
{  
    return joiningDate;  
}  
  
public String getQualification()  
{  
    return qualification;  
}  
  
public String getAppointedBy()  
{  
    return appointedBy;  
}  
  
//boolean is used in conditional statements  
  
public boolean getJoined()  
{  
    return joined;  
}  
  
public boolean getTerminated()  
{  
    return terminated;  
}  
  
//method calling for setting Shift  
  
public void setShift(String newShift)  
{
```

```
        if(joined==false)

        {

            System.out.println("The shift has been changed from " +shift+" to " +newShift+".");

            this.shift=newShift;

        }else{

            System.out.println("The shift can't be changed as the staff has already been
hired.");

        }

    }

    //calling the method by setting the conditional statement of joined

    public void hirePartTimeStaff(String staffName,String joiningDate,String
qualification,String appointedBy)

    {

        // if... else conditional is used to specify a new condition to test, if the first condition
is false

        if(joined==true)

        {

            System.out.println("The staff "+ staffName+ " has already been hired from "+
joiningDate);

            this.terminated=false;

        }else{

            this.staffName=staffName;

            this.joiningDate=joiningDate;

            this.qualification=qualification;
```

```
this.appointedBy=appointedBy;

this.joined=true;

this.terminated=false;

System.out.println("The Staff "+ staffName+ " has been hired for a part-time job.");

}

}

////calling the method by setting the conditional statement of terminated

public void terminateStaff()

{

    if(terminated==true)

    {

        System.out.println("No records of the staff can be found");

    }else{

        System.out.println(getStaffName()+" has been removed from the job");

        this.staffName=" ";

        this.joiningDate=" ";

        this.qualification=" ";

        this.appointedBy=" ";

        this.joined=false;

        terminated=true;

    }

}

}

//Showing the output
```

```
public void displayPartTime()
{
    super.displayInfo();
    if(joined==true)
    {
        System.out.println(".....");
        System.out.println("Vacancy Number: "+getVacancyNumber());
        System.out.println("Designation: " + getDesignation());
        System.out.println("Job Type: " + getJob_Type());
        System.out.println("Staff Name: " +getStaffName());
        System.out.println("Wages per Hour: " + getWagesPerHour());
        System.out.println("Working Hour: " + getWorkingHour());
        System.out.println("Joined Date: " + getJoiningDate());
        System.out.println("Qualification: " + getQualification());
        System.out.println("Appointed By: " + getAppointedBy());
        System.out.println("Income per Day: " + (wagesPerHour*workingHour));
        System.out.println(".....");
    }else{
        System.out.println("No staff has been hired yet for part time job");
    }
}
}
```

- fullTimeStaffHire

//extends keyword is used to indicate that a new class is derived from the base class using inheritance

```
public class fullTimeStaffHire extends staffHire
```

```
{
```

// private is used only for visible within the class, not from any other class(including subclasses)

```
    private int salary;
```

```
    private int workingHour;
```

```
    private String staffName;
```

```
    private String joiningDate;
```

```
    private String qualification;
```

```
    private String appointedBy;
```

```
    private boolean joined;
```

//constructor donot use void because constructor returns the object it creates not void.

```
    public fullTimeStaffHire(int vacancyNumber, String designation, String jobType,int salary, int workingHour)
```

```
    {
```

```
        super(vacancyNumber,designation,jobType);// invoking the super class by using super keyword
```

```
        this.salary=salary;//this keyword is used to initiate current class constructor
```

```
        this.workingHour=workingHour;
```

```
        staffName="";
```

```
        joiningDate="";  
        qualification="";  
        appointedBy="";  
        joined=false;  
    }  
  
    //getter is a method that reads value of a variable  
  
    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;  
}  
  
//boolean is used in conditional statements  
  
public boolean getJoined()  
{  
    return joined;  
}  
  
//method that updates value of a variable  
  
public void setSalary (int newSalary)  
{  
    // if... else conditional is used to specify a new condition to test, if the first condition  
    is false  
  
    if(joined==false)  
    {  
        this.salary= newSalary;  
        System.out.println(" New salary is set to Rs. "+ newSalary+".");  
    }else{
```



```
        System.out.println("It is not possible to change the Salary as staff is already
hired.");
    }
}

//method for setting workingHour

public void setWorkingHour(int newWorkingHour)
{
    if(joined==false)
    {
        this.workingHour= newWorkingHour;

        System.out.println("New Working hour has been set to " +newWorkingHour+"
Hours.");
    }else{
        System.out.println("It is not possible to change the Working Hour as the staff has
already been hired");
    }
}

// creating methods which do not return a value

public void hireFullTimeStaff(String StaffName, String JoiningDate, String Qualification,
String AppointedBy)
{
    if(joined==false)
    {
```

```
        System.out.println("The staff " + StaffName + " is hired by " + AppointedBy + " for  
full-time job.");
```

```
        this.staffName=StaffName;
```

```
        this.joiningDate=JoiningDate;
```

```
        this.qualification=Qualification;
```

```
        this.appointedBy=AppointedBy;
```

```
        this.joined=true;
```

```
    }else{
```

```
        System.out.println("Sorry!! The staff has already been hired.");
```

```
        this.joined=true;
```

```
    }
```

```
}
```

```
//Showing the information that we entered for full time staff hire!!!
```

```
public void displayInfo()
```

```
{
```

```
    super.displayInfo();
```

```
    if(joined==true)
```

```
    {
```

```
        System.out.println(".....");
```

```
        System.out.println("The Name of the Staff is: " + getStaffName()+" .");
```

```
        System.out.println("The Salary of the Staff is: " + getSalary()+" Rupees.");
```

```
        System.out.println("The Working hours of the Staff is: " + getWorkingHour()+"  
hours.");
```

```
        System.out.println("The Joined Date of the Staff is: "+ getJoiningDate()+" .");  
        System.out.println("The qualification of the Staff is: "+ getQualification()+" .");  
        System.out.println("The Staff is appointed by: "+ appointedBy+" .");  
        System.out.println(".....");  
    }else{  
        System.out.println("The Staff has not hired yet for full-time job.");  
    }  
}  
}
```