



# OBJECT ORIENTED PROGRAMMING

ASSIGNMENT

September 2016

Sydney Twigg  
M8C3XRSN8

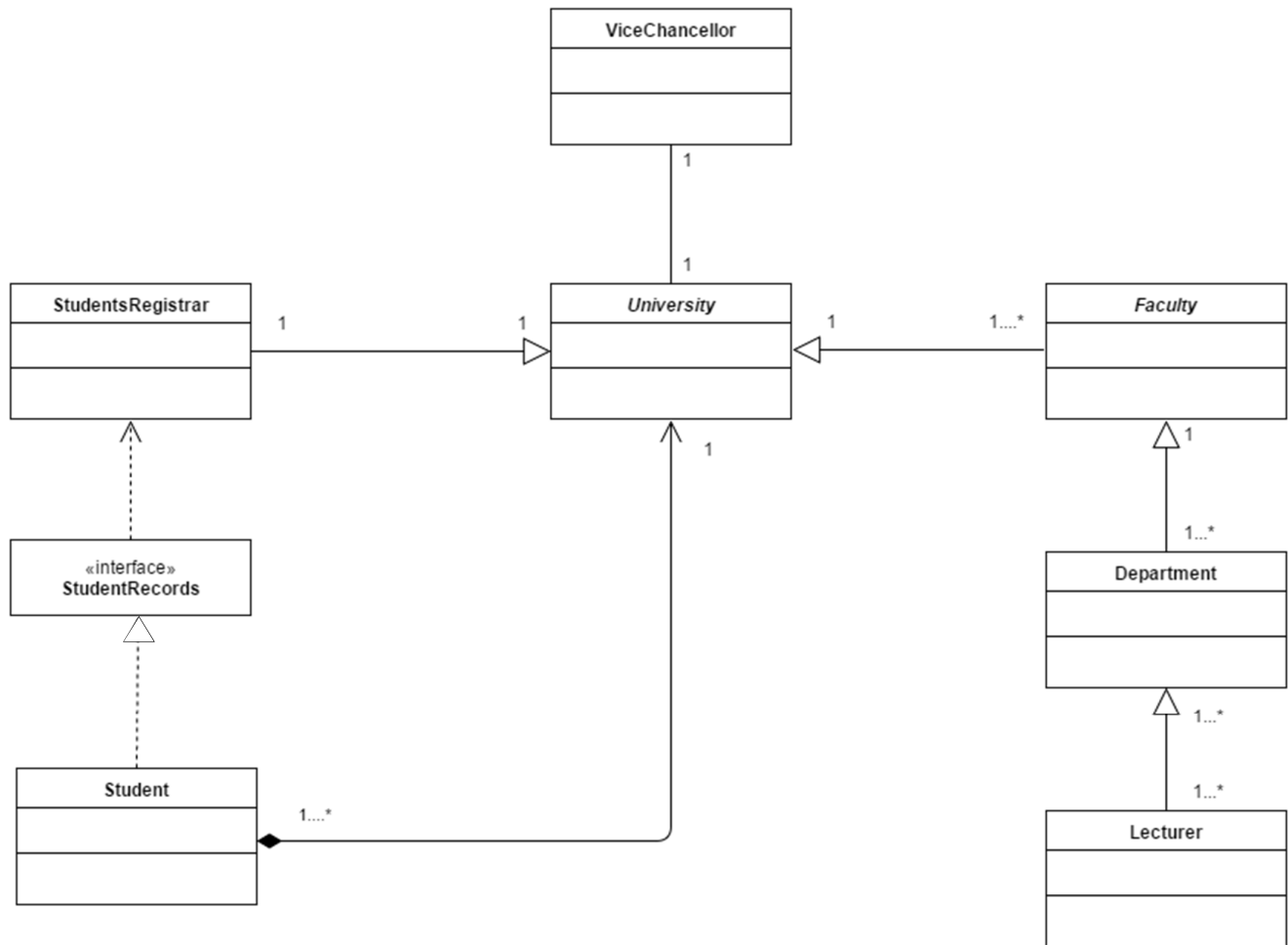
## Contents

QUESTION 1 .....	2
QUESTION 1.1.....	2
QUESTION 1.2.....	2
QUESTION 1.3.....	3
QUESTION 1.4.....	4
QUESTION 1.5.....	4
QUESTION 1.6.....	4
QUESTION 1.7.....	5
QUESTION 2 .....	6
REFERENCES.....	10

## QUESTION 1

### QUESTION 1.1

Class diagram representing the relationships at Rhonda University: (Ivencia, 2016).



Class diagram showing the relationships at Rhonda University

### QUESTION 1.2

CRC diagram for the class Faculty of Rhonda University: (Weisfeld, 2013).

Class name: Faculty	
Responsibilities	Collaborations:
Handle information for and pass information to Departments.	Department. Faculty receives and handles information from University relating to the Departments class, this will then pass that information to Department which inherits from Faculty.
Handle information for and pass information to Lecturers.	Lecturer. Faculty receives and handles information from University relating to the Lecturer class, this will then pass that information to Lecturer which inherits indirectly from Faculty through Department.

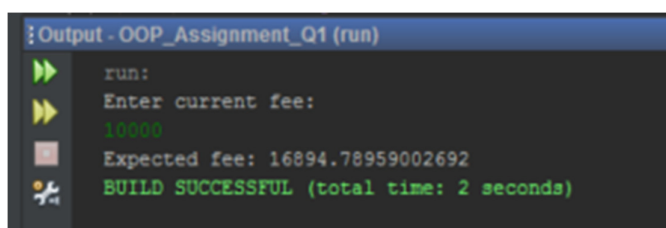
CRC diagram of the Faculty class

## QUESTION 1.3

### TutionFees - java

```
1. //Object Oriented Programming Assignment Question 1.3
2. //Program to calculate and display tution fees in 10 years, with a 6% increase each year
3. //Written by: Sydney Twigg
4. //Last modified: 12/09/2016
5. package pkg;
6.
7. import java.util.Scanner;
8.
9. public class TutionFees {
10.
11.     //Declare variables; currentFee for the 2016 fee & expectedFee for the fee in 2026
12.     private Double currentFee;
13.     private Double expectedFee;
14.
15.     //Constructor for the class, currentFee is initialised in this as it will be inputted
16.     public TutionFees(Double currentFee){
17.         this.currentFee = currentFee;
18.     }
19.
20.     //calculate the fees, and recieve the 2016 fee into the method
21.     public void calcFee(Double currentFee){
22.         //initialise the expected fee to the current fee as a starting point
23.         this.expectedFee = currentFee;
24.
25.         //calculate the expected fee by running it through a for Loop 10 times, adding the 6% each time to the previous
26.         number
27.         for (int i = 0; i < 9; i++){
28.             this.expectedFee = expectedFee + (expectedFee * 0.06);
29.
30.             //test calculation - used to test the output
31.             //System.out.println();
32.             //System.out.println(this.expectedFee);
33.         }
34.
35.         //output method to display the fee in 2026
36.         public void displayAll(){
37.             System.out.println("Expected fee: " + expectedFee);
38.         }
39.
40.         public static void main(String [] args){
41.             //create a variable to hold the user inputted fee
42.             Double currentFee;
43.
44.             //get the user inputted fee
45.             Scanner sc = new Scanner(System.in);
46.             System.out.println("Enter current fee: ");
47.             currentFee = sc.nextDouble();
48.
49.             //Instantiate an object of the TutionFees class and call the constructor
50.             TutionFees tf = new TutionFees(currentFee);
51.
52.             //call the methods from the class to run them
53.             tf.calcFee(currentFee);
54.             tf.displayAll();
55.         }
56.     }
57. //end class
```

Screenshot of TutionFees program. Code available at [codetidy.com/9053/](https://codetidy.com/9053/).



```
Output - OOP_Assignment_Q1 (run)
run:
Enter current fee:
10000
Expected fee: 16894.78959002692
BUILD SUCCESSFUL (total time: 2 seconds)
```

Screenshot showing input and output features of the TutionFees program.

#### QUESTION 1.4

An interface in Java is similar to a class, it is a collection of abstract methods; a class implements an interface thus inheriting all abstract methods from the interface (TutorialsPoint, 2016).

An interface is written similarly to a class, however instead of describing the attributes and behaviours of an object the interface contains behaviours that are implemented by a class. Interfaces cannot be instantiated in Java, thus interfaces do not have constructors and cannot have a method with a body (Kumar, 2016). In an interface, the declared methods are abstract and public by default, and the variables inside an interface are public, static and final by default. Interfaces are used to achieve full abstraction – hiding the irrelevant details from the user – compared to abstract classes which only achieve partial abstraction (Singh, 2013).

An interface has to be implemented by a class before it can be used, the class that implements the interface must implement all methods from the interface. An interface cannot implement another interface; however, an interface can extend other interfaces but they cannot extend classes. A class implements an interface, and an interface extends another class. Java does not support multiple inheritance, however, it can be done using interfaces as a class can implement multiple interfaces. When a class is implementing two or more of the same methods from two different interfaces, only one method needs to be implemented; however, a class cannot implement two methods of the same name but different return types from interfaces (Singh, 2013).

The advantages of using interfaces are that interfaces allow multiple inheritance to be achieved in java and interfaces provide the security of implementation without having to use implementation (Singh, 2013).

The disadvantages of using interfaces are that when designing a project methods need to be chosen carefully as you cannot add or remove methods from an interface later on in the project – this will cause an error in every class that implements the interface; another disadvantage is that if a class that implements the interface has its own methods they cannot directly be used in the code as the type of object is an interface that does not have those methods (Kumar, 2016).

#### QUESTION 1.5

In inheritance, a child class cannot access the private variables inherited from the parent class (University of Maryland, n.d.). This is due to encapsulation – it is assumed that you have not written all classes in the program hence why the restrictions are in place. If the author of the parent class makes changes to the class, the object user will not be aware of the internal changes which could lead to errors when trying to access the parent classes instance variables. Due to the restrictions on private instance variables, any internal changes in the parent class will not affect the child classes code, thus you will not have to change the child class code if you modify the parent class code (University of Maryland, n.d.).

#### QUESTION 1.6

If a developer wishes to modify instance variables that are inherited from the parent class they must use public methods to read and modify the variables from the parent class, as child classes can access public methods and the methods should not be modified in any future updates to the parent class (University of Maryland, n.d.).

## QUESTION 1.7

An example of using public methods to read and modify instance variables from an inherited parent class:

```
//child class squareSize inherits from parent class Square
public class squareSize extends Square {
    //create method to double the length of the square
    public void doubleSquareLength()
    {
        //setLength() and getLength() are public methods from class Square, use these to
        //modify and read the length of the sides of the square in child class squareSize.
        setLength(2 * getLength());
    }
} //end class
```

Adapted from the example in the following: (University of Maryland, n.d.).

## QUESTION 2

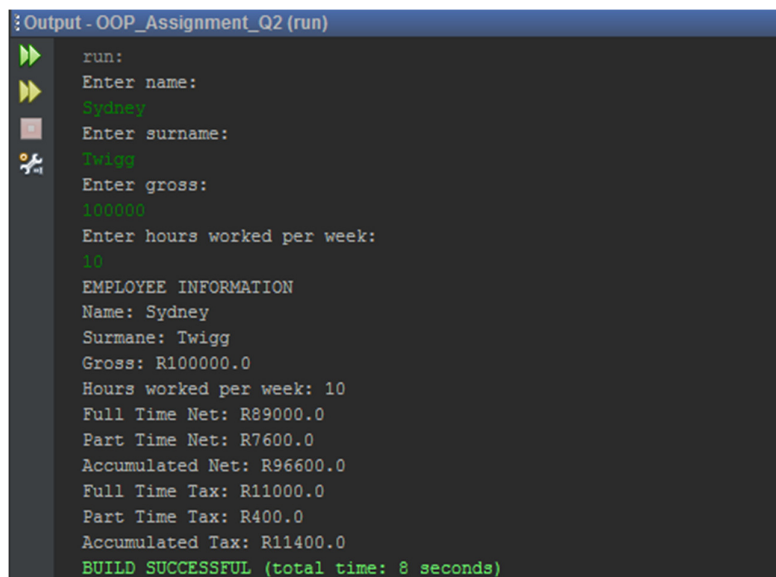
Employee class:

### Employee - java

```
1. //Object Oriented Programming Assignment Question 2
2. //This program calculates the payment structure of full time and part time lecturers, to
   output the part time net pay & tax, full time net pay & tax, and accumulated net pay & tax.
3. //Written by: Sydney Twigg
4. //Last modified: 14/09/2016
5. package pkg;
6.
7. //Abstract class that FullTimeEmployee and PartTimeEmployee will inherit from
8. abstract public class Employee {
9.     //declare variables that will be used throughout the program
10.    String name;
11.    String surname;
12.    double gross;
13.
14.    //abstract method to calculate net salary
15.    abstract double CalculateNet();
16.
17.    //abstract method to calculate tax
18.    abstract double CalculateTax();
19.
20.    //create a constructor in order to access the variables in the child classes, using the
    super function
21.    public Employee(String name, String surname, double gross){
22.        this.name = name;
23.        this.surname = surname;
24.        this.gross = gross;
25.    }
26. }
27. //end class
```

Screenshot of Employee class. Code available at [codetidy.com/9049/](https://codetidy.com/9049/)

Example showing the program functionality:



```
:Output - OOP_Assignment_Q2 (run)
run:
Enter name:
Sydney
Enter surname:
Twigg
Enter gross:
100000
Enter hours worked per week:
10
EMPLOYEE INFORMATION
Name: Sydney
Surname: Twigg
Gross: R100000.0
Hours worked per week: 10
Full Time Net: R89000.0
Part Time Net: R7600.0
Accumulated Net: R96600.0
Full Time Tax: R11000.0
Part Time Tax: R400.0
Accumulated Tax: R11400.0
BUILD SUCCESSFUL (total time: 8 seconds)
```

Screenshot displaying input and output features of the program

Full Time Employee class:

## FullTimeEmployee - java

```
1. //Object Oriented Programming Assignment Question 2
2. //This program calculates the payment structure of full time and part time lecturers, to output the part time net pay & tax,
   full time net pay & tax, and accumulated net pay & tax.
3. //Written by: Sydney Twigg
4. //Last modified: 14/09/2016
5. package pkg;
6.
7. //Class that part time employee will inherit from, this calculates the full time employee tax and net pay
8. public class FullTimeEmployee extends Employee{
9.
10.     //constructor class - using super to recieve name, surname and gross from Employee
11.     public FullTimeEmployee(String name, String surname, Double gross) {
12.         super(name, surname, gross);
13.     }
14.
15.     //accessors
16.     public String getName() {
17.         return name;
18.     }
19.
20.     public String getSurname() {
21.         return surname;
22.     }
23.
24.     public Double getGross() {
25.         return gross;
26.     }
27.
28.     //mutators
29.     public void setName(String name) {
30.         this.name = name;
31.     }
32.
33.     public void setSurname(String surname) {
34.         this.surname = surname;
35.     }
36.
37.     public void setGross(Double gross) {
38.         this.gross = gross;
39.     }
40.
41.     //This method is to calculate the employees tax on their gross pay
42.     Double calculateTax(){
43.         Double tax;
44.         tax = getGross() * 0.11;
45.         return tax;
46.     }
47.
48.     //This method is to calculate the employees net based on gross and tax as calculated above.
49.     Double calculateNet(){
50.         Double tax = calculateTax();
51.         Double net;
52.         net = getGross() - tax;
53.         return net;
54.     }
55.
56.     //abstract method from Employee, has no functionality as it is only allowing the PartTimeEmployee class to inherit from
   Employee through FullTimeEmployee
57.     @Override
58.     double CalculateNet() {
59.         throw new UnsupportedOperationException("Not supported yet.");
60.     }
61.
62.     //abstract method from Employee, has no functionality as it is only allowing the PartTimeEmployee class to inherit from
   Employee through FullTimeEmployee
63.     @Override
64.     double CalculateTax() {
65.         throw new UnsupportedOperationException("Not supported yet.");
66.     }
67. }
68. //end class
```

Screenshot of FullTimeEmployee class. Code available at [codetidy.com/9050/](https://codetidy.com/9050/).



## PartTimeEmployee - java

```

1. //Object Oriented Programming Assignment Question 2
2. //This program calculates the payment structure of full time and part time lecturers, to output the part time net pay &
   tax, full time net pay & tax, and accumulated net pay & tax.
3. //Written by: Sydney Twigg
4. //Last modified: 14/09/2016
5. package pkg;
6.
7. import java.util.Scanner;
8.
9. //Inherits directly from FullTimeEmployee and indirectly from Employee through FullTimeEmployee. This class calculates the
   part time net pay and tax, as well as accumulated net pay and tax.
10. //Main method is in this class, thus this class has the input and display functionality for the program.
11. public class PartTimeEmployee extends FullTimeEmployee {
12.
13.     //Declare and initialise variables. hourlyRate set at R200 as specified in question.
14.     private Double hourlyRate = 200.0;
15.     //Variable to hold the number of hours worked by the employee per week, this allows for part time pay calculations.
16.     private int hoursWorked;
17.
18.     //Constructor to receive the variables from the parent classes using super, as well as to receive the hoursWorked
   variable.
19.     public PartTimeEmployee(String name, String surname, Double gross, int hoursWorked) {
20.         super(name, surname, gross);
21.         this.hoursWorked = hoursWorked;
22.     }
23.
24.     //This allows for the hoursWorked variable to be used in other methods.
25.     public int getHoursWorked() {
26.         return hoursWorked;
27.     }
28.
29.     //Calculate the monthly part time salary based on hours worked per week.
30.     public Double partTimeSalary(){
31.         //hoursWorked is received using the getHoursWorked accessor.
32.         int hoursWorked = getHoursWorked();
33.
34.         // * 4 to make it monthly pay
35.         Double partTimeSalary = (hoursWorked * hourlyRate) * 4;
36.
37.         return partTimeSalary;
38.     }
39.
40.     //inherited abstract method, this calculates the tax on the part time pay earned.
41.     @Override
42.     double CalculateTax(){
43.         //hoursWorked is received using the getHoursWorked accessor.
44.         int hoursWorked = getHoursWorked();
45.         Double tax;
46.         Double taxPerc = 0.0;
47.         //received from the method to calculate the part time salary and stored as a variable.
48.         Double partTimeSalary = partTimeSalary();
49.
50.         //if-else statement to determine the tax percentage based on how many part time hours they work per week.
51.         if (hoursWorked <= 10){
52.             taxPerc = 0.05;
53.         }
54.         else if (hoursWorked > 10){
55.             taxPerc = 0.075;
56.         }
57.
58.         //calculate the tax amount based on the outcome of the if-else statement
59.         tax = partTimeSalary * taxPerc;
60.         return tax;
61.     }
62.

```

```

63. //inherited abstract method, this calculates the employees part time net pay.
64. @Override
65. double CalculateNet() {
66.     //recieve the calculated tax amount from the CalculateTax method and stored as a variable
67.     Double tax = CalculateTax();
68.     //receive the calculated part time salary and stored as a variable
69.     Double partTimeSalary = partTimeSalary();
70.
71.     //calculate part time net
72.     Double net = partTimeSalary - tax;
73.
74.     return net;
75. }
76.
77. //method to calculate accumulated tax, the sum of the part time and full time tax.
78. public Double accumulatedTax() {
79.     //recieve full time tax from the FullTimeEmployee class and store as a variable.
80.     Double fullTimeTax = super.calculateTax();
81.     //recieve part time tax from CalculateTax method and store as a variable.
82.     Double partTimeTax = CalculateTax();
83.
84.     //calculate the sum of the part time and full time tax
85.     Double accumulatedTax = fullTimeTax + partTimeTax;
86.
87.     return accumulatedTax;
88. }
89.
90. //method to calculate the sum of the part time and full time net pay
91. public Double accumulatedNet() {
92.     //recieve full time net from the FullTimeEmployee class and store as a variable
93.     Double fullTimeNet = super.calculateNet();
94.     //recieve part time net from CalculateNet method and store as a variable
95.     Double partTimeNet = CalculateNet();
96.
97.     //calculate the sum of the part time and full time net pay
98.     Double accumulatedNet = fullTimeNet + partTimeNet;
99.
100.    return accumulatedNet;
101. }
102.
103. //method to display all information on the employee
104. public void displayAll(){
105.     //display
106.     System.out.println("EMPLOYEE INFORMATION");
107.     System.out.println("Name: " + name);
108.     System.out.println("Surname: " + surname);
109.     System.out.println("Gross: R" + gross);
110.    //amounts are recieved from methods in this class as well as methods from FullTimeEmployee using the super
function
111.    System.out.println("Hours worked per week: " + getHoursWorked());
112.    System.out.println("Full Time Net: R" + super.calculateNet());
113.    System.out.println("Part Time Net: R" + CalculateNet());
114.    System.out.println("Accumulated Net: R" + accumulatedNet());
115.    System.out.println("Full Time Tax: R" + super.calculateTax());
116.    System.out.println("Part Time Tax: R" + CalculateTax());
117.    System.out.println("Accumulated Tax: R" + accumulatedTax());
118. }
119.
120. //main method
121. public static void main(String [] args){
122.     //Declare variables that will be used to capture inputted data
123.     String name, surname;
124.     Double gross;
125.     int hoursWorked;
126.
127.     //instantiate scanner
128.     Scanner sc = new Scanner(System.in);
129.
130.     //get user inputs
131.     System.out.println("Enter name: ");
132.     name = sc.next();
133.     System.out.println("Enter surname: ");
134.     surname = sc.next();
135.     System.out.println("Enter gross: ");
136.     gross = sc.nextDouble();
137.     System.out.println("Enter hours worked per week: ");
138.     hoursWorked = sc.nextInt();
139.
140.     //instantiate an object of the PartTimeEmployee class and call the constructor
141.     PartTimeEmployee obj = new PartTimeEmployee(name, surname, gross, hoursWorked);
142.
143.     //call the displayAll method to display outputs
144.     obj.displayAll();
145. }
146. }
147. //end class

```

## REFERENCES

Code Tidy, 2016. *Code Tidy*. [Online]

Available at: <https://codetidy.com/>

[Accessed September 2016].

Ivencia, 2016. *UML 4. Class diagrams*. [Online]

Available at: [http://www.ivencia.com/softwarearchitect/chapter1/chapter1\\_6.htm](http://www.ivencia.com/softwarearchitect/chapter1/chapter1_6.htm)

[Accessed September 2016].

Kumar, P., 2016. *Interface in Java*. [Online]

Available at: <http://www.journaldev.com/1601/interface-in-java>

[Accessed September 2016].

Singh, C., 2013. *Interface in java with example programs*. [Online]

Available at: <http://beginnersbook.com/2013/05/java-interface/>

[Accessed September 2016].

TutorialsPoint, 2016. *Java - Inheritance*. [Online]

Available at: [http://www.tutorialspoint.com/java/java\\_interfaces.htm](http://www.tutorialspoint.com/java/java_interfaces.htm)

[Accessed September 2016].

University of Maryland, n.d. *Incremental Java*. [Online]

Available at: <https://www.cs.umd.edu/~clin/MoreJava/Objects/inheritance.html>

[Accessed September 2016].

Weisfeld, M., 2013. *The Object Oriented Thought Process*. 4th ed. s.l.:Addison-Wesley.