Name: SATYAJIT GHANA Registration Number: 17ETCS002159

Laboratory 4

Title of the Laboratory Exercise: Overloading, Inheritance and Overriding

1. Introduction and Purpose of Experiment

   Students apply object oriented programming concepts including Overloading, Inheritance and Overriding to solve problems.

2. Aim and Objectives

   Aim

   To apply object oriented programming concepts including Overloading, Inheritance and Overriding to solve problems

   Objectives

   At the end of this lab, the student will be able to

   - Apply Overloading, Inheritance and Overriding for solving problems
   - Express solutions in Java language
   - Use Netbeans IDE

3. Experimental Procedure

For the problems listed below, design the data structures, algorithm(s) and write the program(s). Tabulate the output for various inputs and verify against expected values. Compare the programming method in Java with C programming languages. Describe your learning along with the limitations of overall approach if any. Suggest how these can be overcome.

a. Write a program to develop a game for the scenario posed:

In ACME organization, there are many employees. All employees have first name, last name and aadhar number. ACME organization creates its own products and sells them. There are two types of sales employees: Commission employee are paid a percentage share (known as commission rate) of their gross sales. Base plus commission employee is a second type of sales employee who is paid a basic salary along with the commission. Other types of Employees include salaried employees who get paid a fixed weekly salary, piece workers who get paid a preset per piece amount based on the

number of pieces they produce and hourly wage employees who get paid an hourly wage. Hourly wage employees also get 1.5 times the hourly wage for hours worked over 40 hours. Create a Java program to calculate salary of an employee in ACME organization.

Name: SATYAJIT GHANA Registration Number: 17ETCS002159

Documentation:	
a. Procedure and Algorithm(s):	**Procedure:** 1. Generating a new Java Application Open Netbeans and create a new Project and select Java -> Java Application, give a proper package name and a proper Project Name, this will generate a CLI Interface Java Application, with a main method in a Class File with the Project's name. 2. Design the Class Diagram Analyze the given problem and identify the classes and objects, find the common state and action of the various objects and classes, basically perform an object decomposition and make a UML Diagram for the same, this will make the development easier while implementing the application. 3. Creating the Classes To separate the models that were identified in the UML Diagram, create a sub-package "models", add the identified classes here. Use the keyword `extends` to inherit a parent class into the base class. Add the classes as per the UML Diagram. Add the state for each of the classes, i.e. the variables inside each of the classes. Use the keyword abstract to declare an abstract class or an abstract method, any class containing an abstract method must be an abstract class. Objects of abstract classes cannot be instantiated. 4. Implementing the methods Now that all the classes have been added, the methods in those needs to be implemented as per the logic for the individual classes, refer the question for the same. Follow the following procedure when implementing the methods for the state inside the class. i. Right click -> insert code -> Constructor : to add the constructor to the class. This method is called whenever an object of this class or its child class is made.

ii. Right click -> insert code -> getters and setters : make sure to click on encapsulate fields, this will make the variables in the class to private, this is required for abstraction. Getters and Setters are methods that can change the state of the object and fetch the current state of the object.

Now the abstract method from the parent classes are to be implemented in the child classes. Write your business logic here in those methods wherever required.

Write the definition for the main method, and make sure to import the previously defined classes using the correct package name.

5. Overriding generic Methods

Java Classes by default inherit the Object Class which comes with some generic methods such as `toString`, `equals`, these can be overridden too. To indicate an overridden method, use the annotation `@Override` before the method definition. This needs to be done to display an Employee on the console.

6. Execute and Debug

Execute the program by clicking on Clean and Build and then Run, and also perform proper tests on it. Verify that the program is as per the specifications required.

7. Documentation

Write documentation for the methods and Classes implemented in the program, with its usage and parameter, the developer's name and date.

<u>Algorithms:</u>

Algorithm getSalary for CommissionEmployeee

Parameters: None

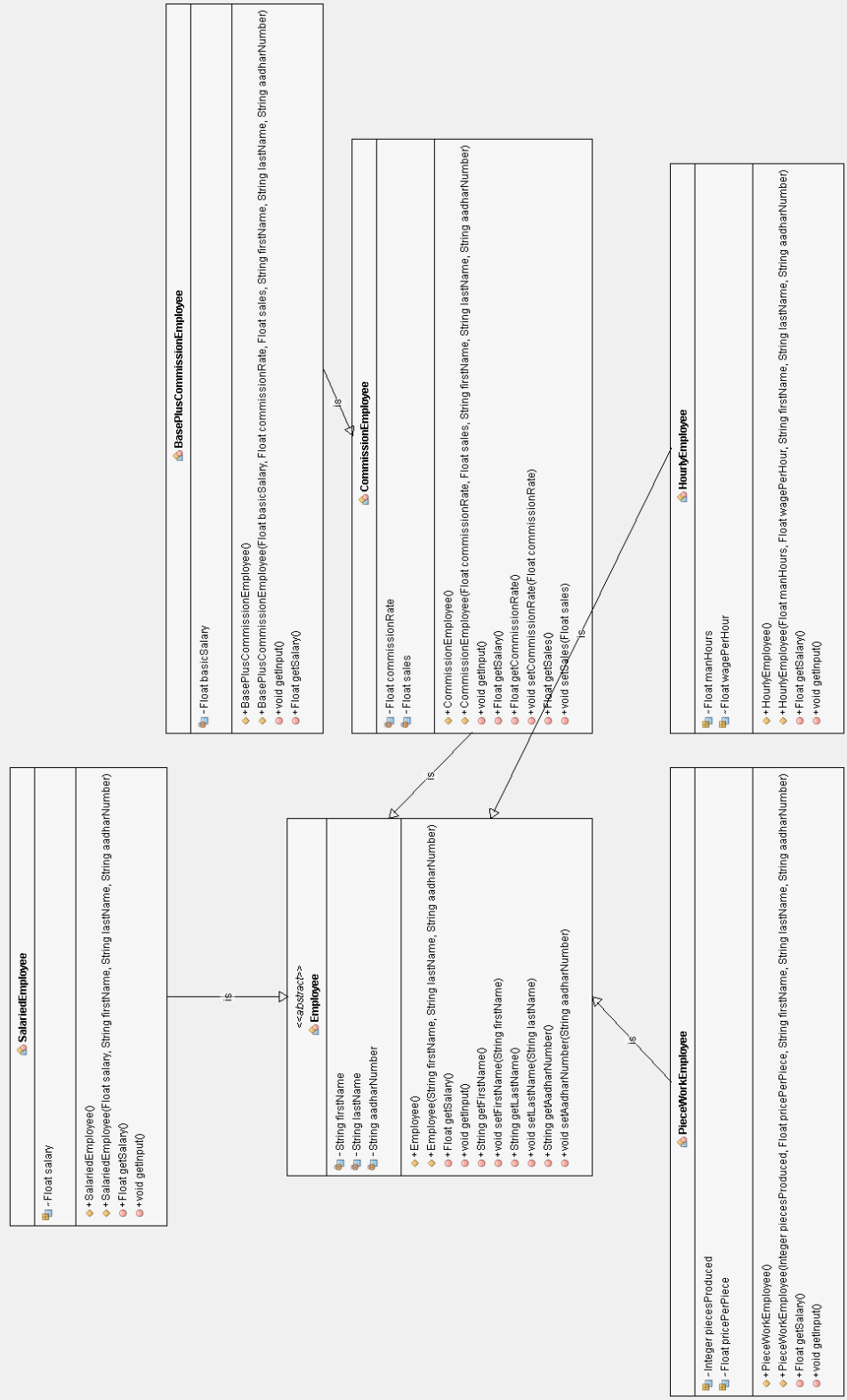
```
Step 1: Start
Step 2: return (commissionRate / 100) * sales
Step 5: Stop
```

Algorithm getSalary for BasePlusCommissionEmployee

	Parameter: None Step 1: Start Step 2: use Algorithm getSalary for CommissionEmployee + basicSalary Step 3: Stop **Algorithm getSalary for HourlyEmployee** **Parameter: None** Step 1: Start Step 2: if manHours >= 40 Step 2.1 return wagePerHour * (40 + (manHours - 40) * 1.5) Step 2.2 Stop Step 3: return manHours * wagePerHour Step 3: Stop **Algorithm getSalary for PieceWorkEmployee** **Parameter: None** Step 1: Start Step 2: return piecesProduced * pricePerPiece Step 3: Stop **Algorithm getSalary for SalariedEmployee** **Parameters: None** Step 1: Start Step 2: return salary Step 3: Stop **Algorithm for main method** **Parameters: Command Line Args, None for this case, except the default Class Name** Step 1: Start Step 2: Display the Menu of Employees Step 3: Instantiate a new Employee and assign null to it Step 4: Take the choice from user and Instantiate the respective employee, and assign it to employee.

	Step 5: If employee != null, then take the input for the employee from user Step 6: Display the employee details Step 7: Stop
b. Conclusions :	Through this lab experiment another core element of Object Oriented Programming was learnt which is Polymorphism, by the usage of Overloading and Overriding in Java. This makes it easy for Code Reuse, the amount of code is reduced and makes it easier to understand for any other developer looking at the same code, basically makes it more intuitive, since we are trying to emulate a real word like objects here it is easier for us humans to relate to it. The getSalary is a common method for all the employees, but the way each type of employee get's it's salary is different hence the definition of getSalary is a little different, this is the overloaded function we are talking about. Apart from polymorphism Inheritance was used since Employee is subtyped into different kinds of employees all of them having a sub-common state, and behavior. For other software engineers looking at this code, it's not easy to debug the code when the engineer does not have the access to the actual code, sure, every class that inherits Employee has to override getSalary, since it's an abstract method, although the grandchild of Employee needn't override it, and uses it's parent's getSalary, so it's not easy to know which getSalary is being called in such a case, I would say this as a disadvantage of taking this route, although a debugger can be used to debug such kind of code and looking at the method calls, again that increases the development process time.

Name: SATYAJIT GHANA Registration Number: 17ETCS002159



Name: SATYAJIT GHANA Registration Number: 17ETCS002159

Results and Discussions:

Screenshot:

```java
public abstract class Employee {
    private String firstName;
    private String lastName;
    private String aadharNumber;

    public Employee() {
    };

    public Employee(String firstName, String lastName, String aadharNumber) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.aadharNumber = aadharNumber;
    }

    public abstract Float getSalary();

    public void getInput() {
        Scanner input = new Scanner(System.in);
        input.useDelimiter("\n");
        System.out.print("Enter First Name : ");
        this.firstName = input.nextLine();
        System.out.print("Enter Last Name : ");
        this.lastName = input.nextLine();
        System.out.print("Enter Aadhar Number : ");
        this.aadharNumber = input.nextLine();
    };

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getAadharNumber() {
        return aadharNumber;
    }

    public void setAadharNumber(String aadharNumber) {
        this.aadharNumber = aadharNumber;
    }
```

Discussion:

Employee is the super class of all the implemented class, which is quite evident from the UML Diagram Draw, it stores the state of the employee such as the `firstName`, `lastName`, and the `aadharNumber`, these are

encapsulated in private fields. Setter methods are implemented to change the state of the Employee and Getter methods are implemented to get the current state of the object.

The Constructor of the Employee class takes the Employee parameters and creates an object with that state. `'this'` is used to refer to the current object of the class.

Since the `getSalary` of the Employee method definition is not known as Employee Class is too generic for it, it is declared as abstract method, so every first child of Employee must define this method, since now Employee has an abstract method, Employee itself should be declared as an abstract class.

`getInput` is a method to read the details of the employee from the console and set that as the state of the object.

Screenshot:

```
public class CommissionEmployee extends Employee {

    private Float commissionRate;
    private Float sales;

    public CommissionEmployee() {
    }

    public CommissionEmployee(Float commissionRate, Float sales, String firstName, String lastName,
String aadharNumber) {
        super(firstName, lastName, aadharNumber);
        this.commissionRate = commissionRate;
        this.sales = sales;
    }

    public void getInput() {
        super.getInput();
        Scanner input = new Scanner(System.in);
        System.out.print("Enter Commission Rate : ");
        this.commissionRate = input.nextFloat();
        System.out.print("Enter Sales : ");
        this.sales = input.nextFloat();
    }

    @Override
    public Float getSalary() {
        return commissionRate/100f * sales;
    }


    public Float getCommissionRate() {
        return commissionRate;
    }

    public void setCommissionRate(Float commissionRate) {
        this.commissionRate = commissionRate;
    }

    public Float getSales() {
        return sales;
    }

    public void setSales(Float sales) {
```

```
        this.sales = sales;
    }

}
```

Discussion:

CommissionEmployee is also an Employee, and hence it inherits the properties of our previously defined Employee, the keyword `extend` is used in java to inherit a class. Since it inherits Employee, it has to call the constructor of its super class, so we do that by using `super(<params>)`, where `<params>` must match the parameters of its super class. All the public methods from the class Employee are inherited into CommissionEmployee and can be used with its object. Since it's parent class is an abstract class and has an abstract method, it needs to be overridden and the method definition must be written, this is done by having a method with the <u>same method signature</u> as the method to be overridden and the annotation `@Override` is used to indicate overridden methods

Screenshot:

```
public class BasePlusCommissionEmployee extends CommissionEmployee {

    private Float basicSalary;

    public BasePlusCommissionEmployee() {
    }

    public BasePlusCommissionEmployee(Float basicSalary, Float commissionRate, Float sales, String
firstName, String lastName, String aadharNumber) {
        super(commissionRate, sales, firstName, lastName, aadharNumber);
        this.basicSalary = basicSalary;
    }

    @Override
    public void getInput() {
        Scanner input = new Scanner(System.in);
        super.getInput();
        System.out.print("Enter basic Salary : ");
        basicSalary = input.nextFloat();
    }

    @Override
    public Float getSalary() {
        return super.getSalary() + basicSalary;
    }

}
```

Discussion:

BasePlusCommissionEmployee is a CommissionEmployee and hence extend it. The constructor of this calls the constructor of its parent class CommssionEmployee which inturns calls its parent class Employee constructor, this is how the object of a BasePlusCommissionEmployee generated. Since it's parent CommissionEmpoyee's getSalary method is already implemented, java will not explicitly complain to override

getSalary, it is then the duty of the programmer to override getSalary, since the salary for this class has a different logic than its parent.

Screenshot:

```java
public class ACMEEmployeeSalary {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);
        String menu =
                "Select the type of employee : \n"+
                "1.\tCommission Employee\n"+
                "2.\tBase Plus Employee\n"+
                "3.\tSalaried Employee\n"+
                "4.\tHourly Employee\n"+
                "5.\tPiece Work Emlpoyee\n"+
                "6.\tExit"+
                "\n\tYour Choice : ";
        System.out.print(menu);
        Integer choice = input.nextInt();
        Employee employee = null;
        switch(choice) {
            case 1:
                employee = new CommissionEmployee();
                break;
            case 2:
                employee = new BasePlusCommissionEmployee();
                break;
            case 3:
                employee = new SalariedEmployee();
                break;
            case 4:
                employee = new HourlyEmployee();
                break;
            case 5:
                employee = new PieceWorkEmployee();
                break;
            case 6:
                System.exit(0);
            default:
                System.out.println("Wrong Choice !");
                break;
        }
        if (employee != null) {
            employee.getInput();
            System.out.println("Your Salary : " + employee.getSalary()+"\n");
        }
        /* Infinite Recursion */
        main(args);
    }

}
```

Discussion:

The main method is the driver method for the ACMEEmployeeSalary, this presents the user with a menu of the kinds of employees supported, the selected employee is created by using the `new` operator that creates

an object of that class and returns a reference to it. The input from the user is taken using the getInput method of employee and the salary for that employee is obtained by calling the getSalary method. getInput method uses the Scanner class of java.

Screenshot:

```
Select the type of employee :
1.      Commission Employee
2.      Base Plus Employee
3.      Salaried Employee
4.      Hourly Employee
5.      Piece Work Emlpoyee
6.      Exit
        Your Choice : 2
Enter First Name : Satyajit
Enter Last Name : Ghana
Enter Aadhar Number : 836498637721
Enter Commission Rate : 12
Enter Sales : 1000
Enter basic Salary : 5000
Your Salary : 5120.0
```

Discussion:

The Application is run using Run option in Netbeans that builds the project and runs it. The user is presented with the menu-drive program, which is also an action-object form of program. The kind of employee is selected using the option, and when selected the respective employee's object is created using the new operator, and the input method is called for that employee by calling the getInput method of the object, the input is taken from the console for that kind of employee and the salary is calculated and displayed on the output console.