Green University of Bangladesh
Department of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering

Semester: (Summer, Year:2021), B.Sc. in CSE (Day)

Lab Report NO #1

Course Title:Object Oriented Programming

Course Code: CSE204 Section:242_d2

Lab Experiment Name: Inheritance.

Student Details

	Name	ID
1.	Tushar Rajbongshi	242002140

Submission Date : 14/12/2025

Course Teacher's Name : Sabbir Hosen Mamun

Lab Report Status

Marks: …… Signature:.....

Comments:..... Date:.....

1. TITLE OF THE LAB REPORT EXPERIMENT:Inheritance.

2.OBJECTIVES:

1. We can understand the concept of inheritance in Java.
2. We can learn how method overriding works in inheritance.
3. We can demonstrate code reusability using parent and child classes.

3. Theory:

Inheritance refers to the process by which one class acquires the properties (data members) and behaviors (methods) of another class. In general terms, inheritance means receiving characteristics, rights, or properties from a predecessor. In Object-Oriented Programming (OOP), it represents the relationship between a parent (base) class and a child (derived) class.In programming, inheritance allows a new class to reuse the existing code of another class. The class from which features are inherited is called the parent class, and the class that inherits those features is called the child class.

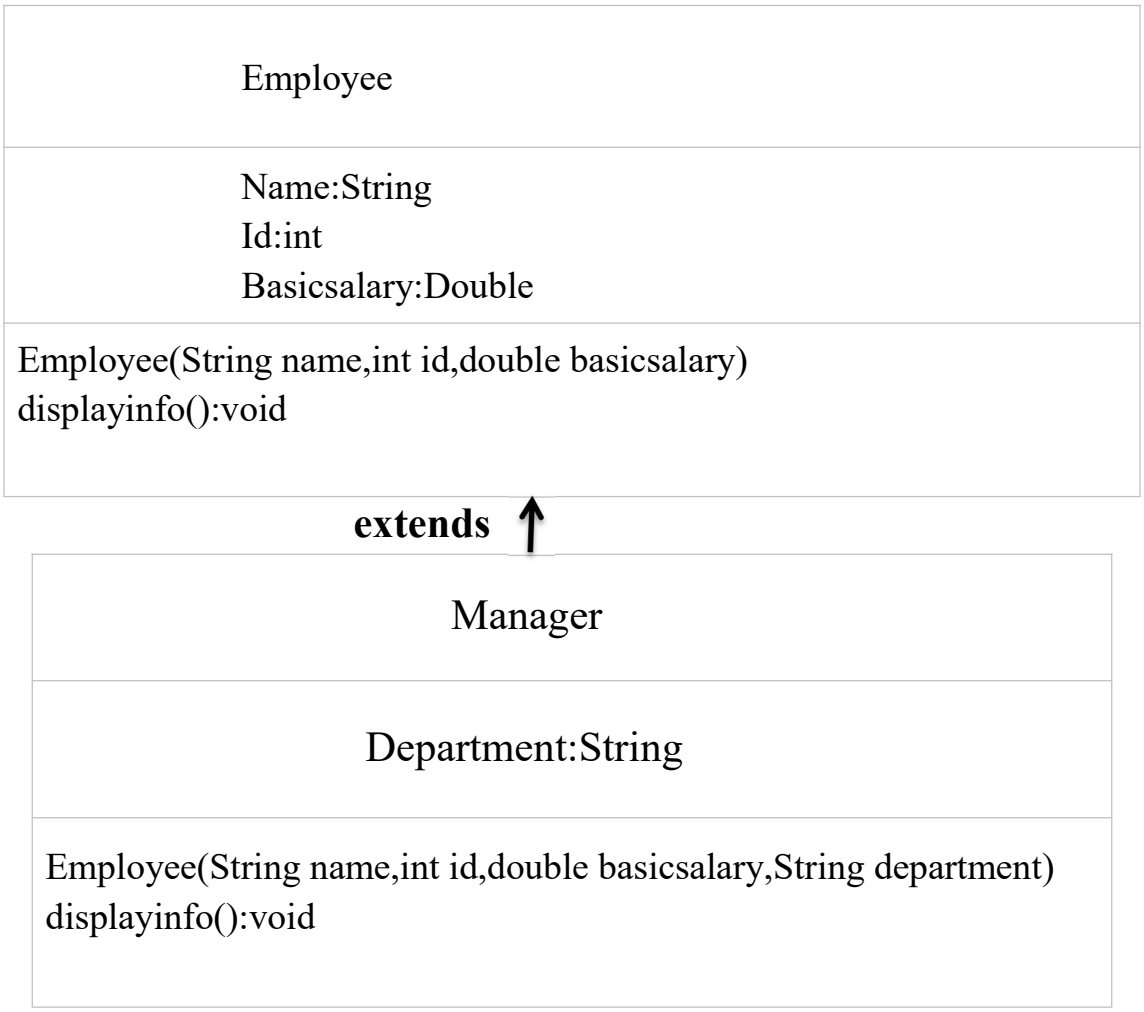
Definition of Inheritance: Inheritance is an OOP mechanism in which a child class inherits data members and methods from a parent class, enabling code reuse and establishing a hierarchical relationship between classes.
Example:For example, if there is an Animal class and a Dog class inherits from it, then the Dog class automatically gains the properties and methods of the Animal class, such as eat() or sleep().

Advantages of Inheritance:

1.Code Reusability
2. Time and Effort Saving
3. Structured and Organized Program
4. Extensibility
5. Data Hiding and Security
6. Reliability

4.PROCEDURE / ANALYSIS / DESIGN :

4.1: UML class diagram:



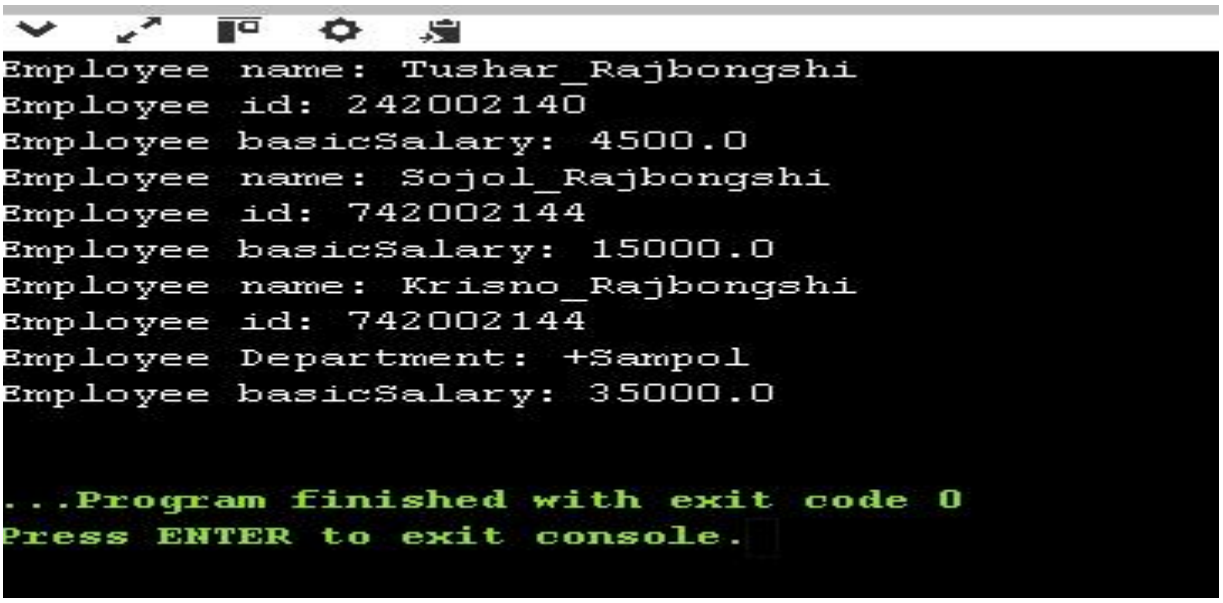
5.IMPLEMENTATION:

```
class Employee
{
    String Employee_name;
    int Employee_id;
    private double Employee_basicSalary;

    public Employee(String name,int id,double basicsalary)
    {
        this.Employee_name=name;
        this.Employee_id=id;
        this.Employee_basicSalary=basicsalary;
    }

    public void displayInfo()
    {
        System.out.println("Employee name: "+this.Employee_name+"\nEmployee id: "+this.Employee_id
            +"\nEmployee basicSalary: "+this.Employee_basicSalary);
    }
    public double getEmployee_basicSalary()
    {
        return this.Employee_basicSalary;
    }
}
```

```
class Manager extends Employee
{
    String Employee_department;

    public Manager(String name,int id,double basicsalary,String department)
    {
        super(name,id,basicsalary);
        this.Employee_department=department;
    }

     public void displayInfo()
    {
        System.out.println("Employee name: "+this.Employee_name+"\nEmployee id: "+this.Employee_id
            +"\nEmployee Department: +"+this.Employee_department+"\nEmployee basicSalary: "+this
            .getEmployee_basicSalary());
    }

}
```

```
     public void displayInfo()
    {
        System.out.println("Employee name: "+this.Employee_name+"\nEmployee id: "+this.Employee_id
            +"\nEmployee Department: +"+this.Employee_department+"\nEmployee basicSalary: "+this
            .getEmployee_basicSalary());
    }

}
public class Inheritance {
    public static void main(String[] args) {

        Employee E1=new Employee("Tushar_Rajbongshi", 242002140, 4500);
        Employee E2=new Employee("Sojol_Rajbongshi", 742002144, 15000);
        Manager M1=new Manager("Krisno_Rajbongshi", 742002144, 35000,"Sampol");
        E1.displayInfo();
        E2.displayInfo();
        M1.displayInfo();


    }

}
```

6.TEST RESULT / OUTPUT:



7. ANALYSIS AND DISCUSSION:

The main objective of this experiment was to understand *Encapsulation* in Java OOP and to apply it practically through a real-life example. For this purpose, a BankAccount class was designed, where all important data members were declared as private. As a result, the internal data of the class cannot be directly accessed or modified from any external class.

During the implementation of the program, it was observed that Encapsulation plays a vital role in ensuring data security and maintaining data integrity. For example, the balance variable cannot be modified directly from the main method. Instead, the balance must be updated through the setBalance() method, where any negative input value is automatically rejected. This demonstrates that Encapsulation ensures data validation and prevents invalid inputs.

Furthermore, the program highlights another important aspect of Encapsulation—*maintainability* and *flexibility*. If the internal logic of the BankAccount class is changed in the future (such as adding transaction history or modifying balance calculation rules), no changes will be required in the external code as long as the structure of the public methods remains unchanged.

In conclusion, the use of Encapsulation has provided the program with a well-organized and modular structure. Each BankAccount object functions as a self-contained unit, making the code easier to understand, debug, and extend in the future. From this analysis, it is clearly evident that Encapsulation is a crucial OOP principle that is essential for developing secure, reliable, and maintainable software.

8. GitHub Repository Link:

https://github.com/Rajbongshi2005/LabReport01-Inheritance.git