# PRACTICAL 7

**Aim**: To understand concepts of Encapsulation and Inheritance in Java programming Language

**Prerequisite:**

* Knowledge of basic constructs in Java programming language.

**Outcome:** After successful completion of this experiment students will be able to,

* To use public and private modifiers
* To use *getter* and *setter* methods
* Develop subclass from a superclass through inheritance

**Theory:**

**Access Modifiers:**

Access modifiers are keywords that can be used to control the visibility of fields, methods, and constructors in a class. The four access modifiers in Java are public, protected, default, and private.

**Types of Access Modifiers:**

* **Private**: We can access the **private modifier**only within the same class and not from outside the class.
* **Default:** We can access the **default modifier**only within the same package and not from outside the package. And also, if we do not specify any access modifier it will automatically consider it as default.
* Protected: We can access the protected modifier within the same package and also from outside the package with the help of the child class. If we do not make the child class, we cannot access it from outside the package. So inheritance is a must for accessing it from outside the package.
* Public: We can access the public modifier from anywhere. We can access public modifiers from within the class as well as from outside the class and also within the package and outside the package.

**Public Access Modifier**

* It is a keyword. If a class member like variable, method, or data members are prefixed with**a public access modifier**, then they can be accessed from anywhere inside the program. That is, they can be accessed within the same class as well as from outside the different classes.
* It also includes access within the same package and also from outside the package. The members like variables, methods, and other data members can be accessed globally.
* Using public access modifiers, we can provide access to the members most simply. There are no restrictions on public access modifier members. Hence, it has the widest accessibility or visibility scope as compared to the rest of the access modifiers.

**Default Access Modifier**

* It is not a keyword. Any Java members such as class or methods or data members when not specified with any access modifier they are by default considered as **default access modifiers.**These methods or data members are **only accessible within the same package** and they cannot be accessed from outside the package. It provides more visibility than a private access modifier. But this access modifier is more restricted than protected and public access modifiers.

**Private Access Modifier**

* The private access modifier is specified when any member of a class is prefixed with the **private keyword.**In comparison with the other access modifiers, this is the most restricted access modifier.
* When the methods or data members are prefixed with a **private**access modifier, the visibility of these methods and data members are restricted so, they can be accessed **only within the same class where they have been declared,**they will not be visible to the outside world.
* If we have another class from the same package still, we will not be able to access these methods or data members. So usually, we keep the class variables and methods as **private**, which are intended to be used inside the same class where declared.

**Getter and Setter Methods:**

Encapsulation is a principle of wrapping data (variables) and code together as a single unit. Frequently, Java encapsulation is referred as **data hiding**. But more than data hiding, encapsulation concept is meant for better management or grouping of related data. To achieve a lesser degree of encapsulation in Java, you can use modifiers like "protected" or "public". With encapsulation, developers can change one part of the code easily without affecting other

If a data member is declared "private", then it can only be accessed within the same class. No outside class can access data member of that class. If you need to access these variables, you have to use public "getter" and "setter" methods.

Example:

class Account{

private int account\_number;

private int account\_balance;

// getter method

public int getBalance() {

return this.account\_balance;

}

// setter method

public void setNumber(int num) {

this.account\_number = num;

}

}

**Inheritance in Java** is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the **IS-A relationship** which is also known as a parent-child relationship.

**Use of inheritance:**

* For Method Overriding (so runtime polymorphism can be achieved).
* For Code Reusability.

**Terms used:**

* **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
* **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
* **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
* **Reusability:** As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

**Syntax:**

**class** Subclass-name **extends** Superclass-name

{

   //methods and fields

 }

The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

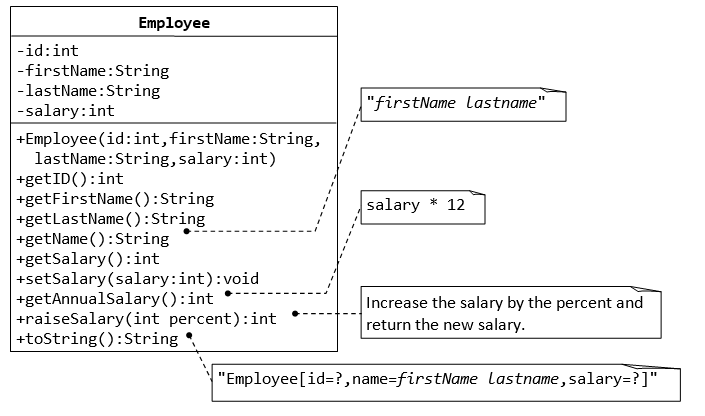
**Types of inheritance in java**

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical. In java programming, multiple and hybrid inheritance is supported through interface only.

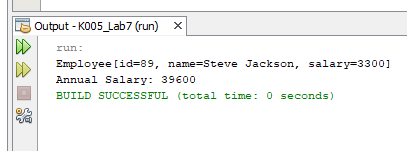
(TO BE COMPLETED BY STUDENTS)

|  |  |
| --- | --- |
| Roll No.: K005 | Name: Jal Bafana |
| Class: Btech. Cyber Security | Batch: K1 |
| Date of Practical: 10.03.2025 | Date of Submission: 10.03.2025 |
| Grade: |  |

1. Design class as shown in the Class diagram below.



Test run the program with employee record **being [89, ”Steve”,”Jackson”,3000]** , the resultant annual employee record with a percent hike of 10% in salary should be **[89, ”Steve”,”Jackson”,396000].**

****

package k005\_lab7;

class Employee {

private int id;

private String firstName;

private String lastName;

private int salary;

// Constructor

public Employee(int id, String firstName, String lastName, int salary) {

this.id = id;

this.firstName = firstName;

this.lastName = lastName;

this.salary = salary;

}

// Getter methods

public int getID() {

return id;

}

public String getFirstName() {

return firstName;

}

public String getLastName() {

return lastName;

}

public String getName() {

return firstName + " " + lastName;

}

public int getSalary() {

return salary;

}

// Setter method

public void setSalary(int salary) {

this.salary = salary;

}

// Get annual salary

public int getAnnualSalary() {

return salary \* 12;

}

// Raise salary by percentage

public int raiseSalary(int percent) {

salary += salary \* percent / 100;

return salary;

}

// toString method

@Override

public String toString() {

return "Employee[id=" + id + ", name=" + getName() + ", salary=" + salary + "]";

}

// Main method to test

public static void main(String[] args) {

Employee emp = new Employee(89, "Steve", "Jackson", 3000);

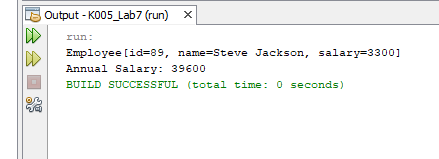
emp.raiseSalary(10);

System.out.println(emp);

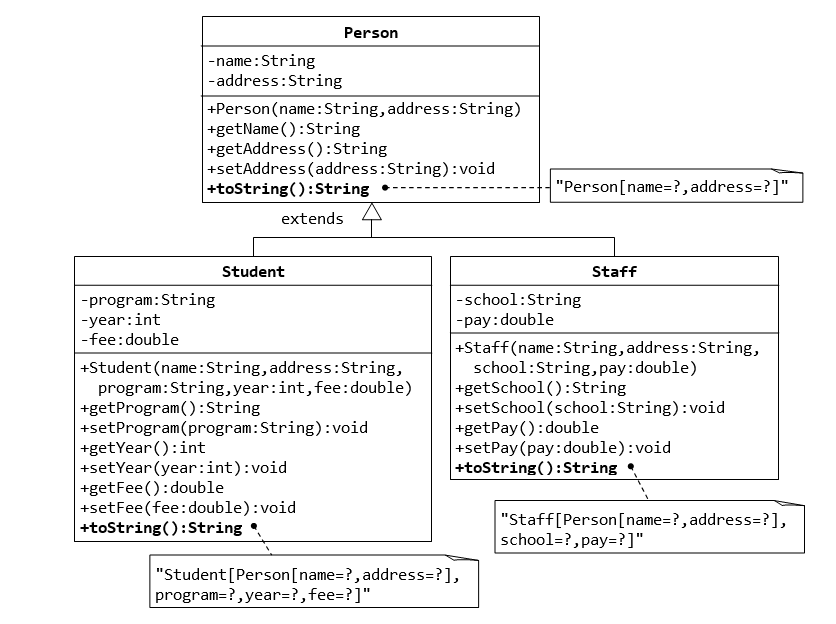
System.out.println("Annual Salary: " + emp.getAnnualSalary());

}

}

****

1. Design and implement classes as shown in the Class diagram below. Write a test program to test the classes created.



package k005\_lab7;

class Person

{

private String name;

private String address;

public Person(String name, String address)

{

this.name = name;

this.address = address;

}

public String getName()

{

return name;

}

public String getAddress()

{

return address;

}

public void setAddress(String address)

{

this.address = address;

}

@Override

public String toString()

{

return "Person[name=" + name + ", address=" + address + "]";

}

}

class Student extends Person

{

private String program;

private int year;

private double fee;

public Student(String name, String address, String program, int year, double fee)

{

super(name, address);

this.program = program;

this.year = year;

this.fee = fee;

}

public String getProgram()

{

return program;

}

public void setProgram(String program)

{

this.program = program;

}

public int getYear()

{

return year;

}

public void setYear(int year)

{

this.year = year;

}

public double getFee()

{

return fee;

}

public void setFee(double fee)

{

this.fee = fee;

}

public String toString()

{

return "Student[" + super.toString() + ", program=" + program + ", year=" + year + ", fee=" + fee + "]";

}

}

class Staff extends Person

{

private String school;

private double pay;

public Staff(String name, String address, String school, double pay)

{

super(name, address);

this.school = school;

this.pay = pay;

}

public String getSchool()

{

return school;

}

public void setSchool(String school)

{

this.school = school;

}

public double getPay()

{

return pay;

}

public void setPay(double pay)

{

this.pay = pay;

}

public String toString()

{

return "Staff[" + super.toString() + ", school=" + school + ", pay=" + pay + "]";

}

}

public class Pgm2

{

public static void main(String[] args)

{

Student student = new Student("Alice", "123 Main St", "Computer Science", 2024, 15000.50);

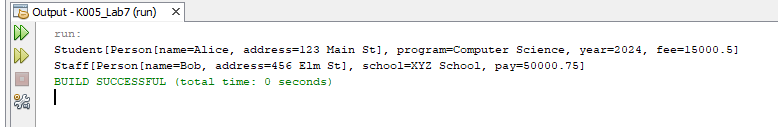
System.out.println(student);

Staff staff = new Staff("Bob", "456 Elm St", "XYZ School", 50000.75);

System.out.println(staff);

}

}

****