# Experiment No. 3

## Title: Implement the concept of Inheritance in Java.

**Batch: SY-IT(B3)** **Roll No.: 16010423076** **Experiment No.:3**

**Aim**: Create a super class Employee having attributes name and ID. Using multilevel inheritance create class Department and Salary. Department class should get the details such as Bonus, Basic pay, garde pay, HRA, TA and DA of the Employee and Salary class should display salary of employee depending on number of days present.

---

**Resources needed: java**

---

**Theory:**

---

Inheritance means to take something that is already made. It is one of the most important feature of Object Oriented Programming. It is the concept that is used for reusability purpose. Inheritance is the mechanism through which we can derived classes from other classes. The derived class is called as child class or the subclass or we can say the extended class and the class from which we are deriving the subclass is called the base class or the parent class. To derive a class in java the keyword extends is used. To clearly understand the concept of inheritance you must go through the following example.

The concept of inheritance is used to make the things from general to more specific e.g. When we hear the word vehicle then we got an image in our mind that it moves from one place to another place it is used for traveling or carrying goods but the word vehicle does not specify whether it is two or three or four wheeler because it is a general word. But the word car makes a more specific image in mind than vehicle, that the car has four wheels . It concludes from the example that car is a specific word and vehicle is the general word. If we think technically to this example then vehicle is the super class (or base class or parent class) and car is the subclass or child class because every car has the features of it's parent (in this case vehicle) class.

The following kinds of inheritance are there in java.

1. Single level Inheritance
2. Multilevel Inheritance
3. Hierarchical Inheritance
4. Hybrid Inheritance

Single level Inheritance

When a subclass is derived simply from it's parent class then this mechanism is known as simple inheritance or single level inheritance. In case of simple inheritance there is only a sub class and it's parent class. It is also called single inheritance or one level inheritance.

eg.

```
class A {
  int x;
  int y;
  int get(int p, int q){
  x=p; y=q; return(0);
```

```
  }
  void Show(){
  System.out.println(x);
  }
}

class B extends A{
  public static void main(String args[]){
  A a = new A();
  a.get(5,6);
  a.Show();
  }
  void display(){
  System.out.println("B");
  }
}
```

Multilevel Inheritance

It is the enhancement of the concept of inheritance. When a subclass is derived from a derived class then this mechanism is known as the multilevel inheritance. The derived class is called the subclass or child class for it's parent class and this parent class works as the child class for it's just above (parent) class. Multilevel inheritance can go up to any number of level.

e.g.

```
class A {
  int x;
  int y;
  int get(int p, int q){
  x=p; y=q; return(0);
  }
  void Show(){
  System.out.println(x);
  }
}
class B extends A{
  void Showb(){
  System.out.println("B");
  }}
class C extends B{
  void display(){
  System.out.println("C");
  }
  public static void main(String args[]){
  A a = new A();
  a.get(5,6);
  a.Show();
```

```
  }
}
```

Hierarchical Inheritance

When more than one subclass is derived from one superclass, then this mechanism is called as hierarchical inheritance.

Eg:

```
class A {
  int x;
  int y;
  int get(int p, int q){
  x=p; y=q; return(0);
  }
  void Show(){
  System.out.println(x);
  }
}

class B extends A{
  void Showb(){
  System.out.println("B");
  }
}

class C extends A{
  void Showc(){
  System.out.println("C");
  }

public static void main(String args[]){
  C c = new C();
  c.get(3,4);
  c.Show();
  c.Showc();
  }
}
```

Hybrid Inheritance

The combination of any of the single level inheritance, multilevel inheritance and hierarchical inheritance is called as hybrid inheritance.

**Multiple Inheritance is not supported in java**

**Access Specifiers:**

It allows the programmer to control the visibility of the class members. Java addresses four categories of visibility for class members

1. public
2. private
3. protected
4. default

public:
When a member of a class is modified by the public specifier, then that member can be accessed by any other class.

private:
When a member of a class is declared as private, then that member can be accessed by only the other members of the same class.

protected:
When a member of a class is declared as protected, then that member can be accessed by only the subclasses of that class.
default:
When a member does not have an explicit access specification, it is visible to subclasses as well as to other classes in the same package. This is the default access.

The table below shows the class member access with respect to packages and subclasses.

| | **private** | **(No modifier) default** | **protected** | **public** |
|---|---|---|---|---|
| Same class | Yes | Yes | Yes | Yes |
| Same package subclass | No | Yes | Yes | Yes |
| Same package non-subclass | No | Yes | Yes | Yes |
| Different package subclass | No | No | Yes | Yes |
| Different package non-subclass | No | No | No | Yes |

**Abstract classes and methods:**
There are situations in which we want to define a superclass that declares the structure of a given abstraction without providing a complete implementation of every method. That is, sometimes we want to create a superclass that only defines a generalized form that will be shared by all of its subclasses, leaving it to each subclass to fill in the details. Such a class determines the nature of the methods that the subclasses must implement. One way this situation can occur is when a superclass is unable to create a meaningful implementation for a method, thus the method is incomplete. Such incomplete methods are called as abstract methods. They are declared as abstract by preceding the method declaration with the abstract keyword ad they do not have method body.

A class having one or more abstract methods is called an abstract class. A class is declared as abstract by preceding the class declaration with the abstract keyword. No objects of the abstract class can be created. Also abstract constructors or abstract static methods cannot be created. Only a reference of an abstract class can be created.

Any subclass of an abstract class must either implement all of the abstract methods in the superclass using method overriding, or be itself declared as abstract.

Eg:

```
abstract class A {
  int x;
  int y;
  int get(int p, int q){
  x=p; y=q; return(0);
  }
  abstract void Show();
}
class B extends A{
  void Show(){
  System.out.println("x="+x" y="+y);
  }
}

class Demo {
  public static void main(String args[]){
  B b = new B();
  b.get(3,4);
  b.Show();

    }
}
```

---

**Results: (Program with output)**

```
class Employee {
    String name;
    int id;

    Employee(String name, int id) {
        this.name = name;
        this.id = id;
    }
}

class Department extends Employee {
    double bonus, basicPay, gradePay, hra, ta, da;

    Department(String name, int id, double bonus, double basicPay, double gradePay, double hra, double ta, double da) {
        super(name, id);
        this.bonus = bonus;
        this.basicPay = basicPay;
        this.gradePay = gradePay;
        this.hra = hra;
```

```
        this.ta = ta;
        this.da = da;
    }
}

class Salary extends Department {

    Salary(String name, int id, double bonus, double basicPay, double gradePay, double hra, double ta, double da) {
        super(name, id, bonus, basicPay, gradePay, hra, ta, da);
    }

    void calculateSalary(int daysPresent) {
        double dailySalary = basicPay + gradePay + hra + ta + da + (bonus/30);
        double totalSalary = dailySalary * daysPresent;
        System.out.println("Salary of Employee " + name + " (ID: " + id + ") is: " + totalSalary);
    }
}

public class Main {
    public static void main(String[] args) {
        Salary emp = new Salary("Ritesh Jha", 101, 3000, 20000, 5000, 8000, 2000, 1500);

        emp.calculateSalary(28);
    }
}
```

**Output :**

Salary of Employee Ritesh Jha (ID: 101) is: 1024800.0

---

**Questions:**

1.Differentiate between method overloading and method overriding with an example program of each.

Method Overloading vs Method Overriding

Overloading: Same method name, different parameters, within the same class.

Example -

```
class MathOperations {
    int add(int a, int b) { return a + b; }
    int add(int a, int b, int c) { return a + b + c; }
}
```

Overriding: Subclass redefines a method from the superclass with the same signature.

Example -

```
class Animal { void sound() { System.out.println("Animal sound"); } }
class Dog extends Animal { void sound() { System.out.println("Dog barks"); } }
```

2. Explain the following uses of the final keyword with an example of each
a) To create a named constant
Example -
final double PI = 3.14159;

b) To prevent a method from being overridden
Example -
class Base { final void show() { System.out.println("Can't override me!"); } }

c) To prevent a class from being inherited
Example -
final class FinalClass { }

---

**Outcomes:**

CO2: Apply string manipulation functions, inheritance, and polymorphism using Java programming.

---

**Conclusion: (Conclusion to be based on the outcomes achieved)**
By this experiment I learned how to implement inheritance in Java by extending classes and utilizing the super keyword to access parent class properties. I also explored method overriding by customizing inherited methods and applying basic salary calculations using object-oriented principles.

---

**Grade: AA / AB / BB / BC / CC / CD /DD**

Signature of faculty in-charge with date

---

**References:**

**Books/ Journals/ Websites:**

1. Herbert Schildt, “Java: The Complete Reference” Tata McGrawHill Publishing Company Limited Tenth Edition, 2017
2. Sachin Malhotra, Saurab Choudhary, “Programming in Java” Oxford University Press Second Edition, 2018 3.
3. D.T. Editorial Services, “Java 8 Programming Black Book” Dream tech Press Edition 2015.