# LAB - 5

## Java Programming Lab

### Topics Covered

Inheritance , Polymorphism , Method overiding

Syeda Reeha Quasar
14114802719
4C7

# EXPERIMENT – 5.1

## Aim:

WAP that implements method overriding.

## Theory:

Method Overriding in Java

- Understanding the problem without method overriding
- Can we override the static method
- Method overloading vs. method overriding

If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in Java.

In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

**Usage of Java Method Overriding**

1. Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
2. Method overriding is used for runtime polymorphism

**Rules for Java Method Overriding**

1. The method must have the same name as in the parent class
2. The method must have the same parameter as in the parent class.
3. There must be an IS-A relationship (inheritance).

**Concepts:**

- **class** keyword is used to declare a class in Java.
- **public** keyword is an access modifier that represents visibility. It means it is visible to all.
- **static** is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method. The main() method is executed by the JVM, so it doesn't require creating an object to invoke the main() method. So, it saves memory.
- **void** is the return type of the method. It means it doesn't return any value.
- **main** represents the starting point of the program.
- **String[] args** or **String args[]** is used for command line argument. We will discuss it in coming section.
- **System.out.println()** is used to print statement. Here, System is a class, out is an object of the PrintStream class, println() is a method of the PrintStream class. We will discuss the internal working of System.out.println() statement in the coming section.
- **Java Utils:** Resources Job Search Discussion. Java. util package contains the **collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes**. This reference will take you through simple and practical methods available in java.
- util. Java util package contains **collection framework, collection classes**, classes related to date and time, event model, internationalization, and miscellaneous utility classes. ... On importing this package, you can access all these classes and methods.
- **Scanner** is a class in **java. util package used for obtaining the input of** the primitive types like int, double, etc. and strings. ... To create an object of Scanner class, we usually pass the predefined object System.in, which represents the standard input stream.
- A switch statement **allows a variable to be tested for equality against a list of values**. Each value is called a case, and the variable being switched on is checked for each case.
- The input is **the data that we give to the program**. The output is the data what we receive from the program in the form of result. Stream represents flow of data or the sequence of data.

## Source Code:

```
class parent{
    void show(){
        System.out.println("In parent class");
    }
}
public class child extends parent{
    void show(){
        System.out.println("In child class");
    }
    public static void main(String[] args){
        child c=new child();
        c.show();
    }
}
```

## Output:

```
In child class
PS C:\Users\Volted User\OneDrive\Desktop\practical>
```

# EXPERIMENT – 5.2

## Aim:

WAP to illustrate simple inheritance.

## Theory:

The process by which one class acquires the properties(data members) and functionalities(methods) of another class is called inheritance. The aim of inheritance is to provide the reusability of code so that a class has to write only the unique features and rest of the common properties and functionalities can be extended from the another class.

**Child Class:**
The class that extends the features of another class is known as child class, sub class or derived class.

**Parent Class:**
The class whose properties and functionalities are used(inherited) by another class is known as parent class, super class or Base class.

**Inheritance** is a process of defining a new class based on an existing class by extending its common data members and methods.
Inheritance allows us to reuse of code, it improves reusability in your java application.
Note: The biggest advantage of Inheritance is that the code that is already present in base class need not be rewritten in the child class.

This means that the data members(instance variables) and methods of the parent class can be used in the child class as.

If you are finding it difficult to understand what is class and object then refer the guide that I have shared on object oriented programming: OOPs Concepts

**Syntax: Inheritance in Java**

To inherit a class we use extends keyword. Here class XYZ is child class and class ABC is parent class. The class XYZ is inheriting the properties and methods of ABC class.

class XYZ extends ABC{}

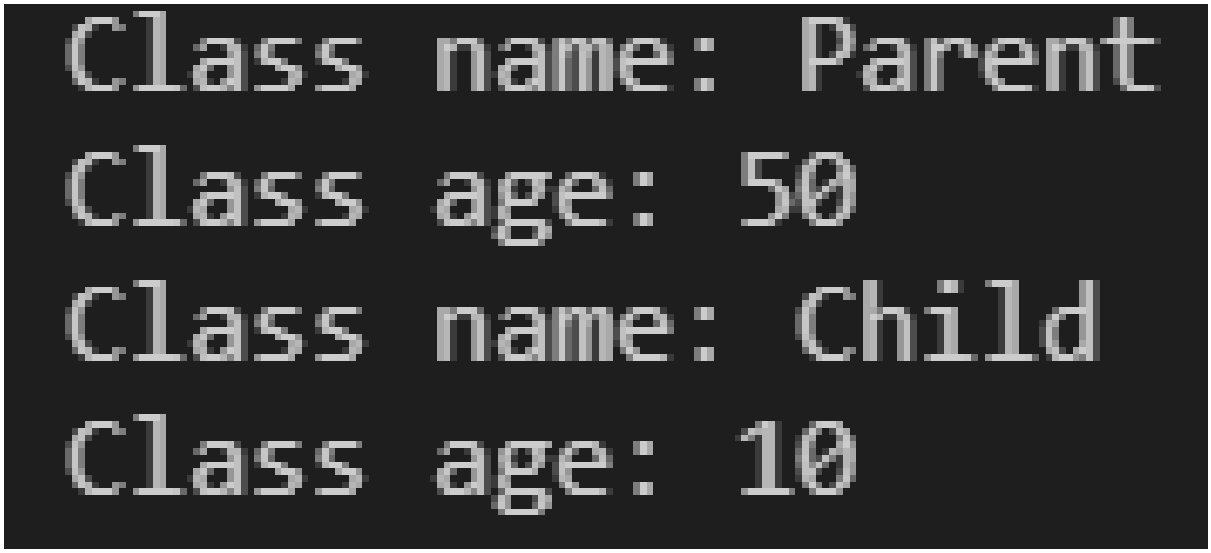
## Source Code:

```
class parents{
    private String str= "Parent";
    private int age=50;
    void displayParent(){
        System.out.println("Class name: "+str);
        System.out.println("Class age: "+age);
    }
}
public class childs extends parents{
    private String str;
    private int age;
    childs(){
        str="undefined";
        age=-1;
    }
    childs(String str,int age){
        this.str=str;
        this.age=age;
    }
    void displayChild(){
        System.out.println("Class name: "+str);
```

```
        System.out.println("Class age: "+age);
    }
    public static void main(String[] args){
        childs c=new childs("Child",10);
        c.displayParent();
        c.displayChild();
    }
}
```

**Output:**

```
Class name: Parent
Class age: 50
Class name: Child
Class age: 10
PS C:\Users\Volted User\OneDrive\Desktop\practical>
```



# EXPERIMENT – 5.3

## Aim:

WAP to illustrate multilevel inheritance.

## Theory:

- **Class :** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical. It represents the set of properties or methods that are common to all objects of one type. A class in java has its methods, variables.

- **Object :** It is a basic unit of Object-Oriented Programming and represents the real life entities. A typical Java program creates many objects, interact by invoking methods. We create object of a class in another class to access its methods and variables. We can declare an object by using the Class Name of the class of which we want to create an object, followed by object name, e.g Circle c1; . An object can access variables of other class by using the . operator , e.g c1.radius , and invoke other class methods by using the method name and passing the parameters(if required), e.g c1.getArea(radius);

- **New Operator :** The new operator instantiates a class by allocating memory for a new object and returning a reference to that memory. The new operator also invokes the class constructor. Example of object initialisation : Circle c1 = new Circle();

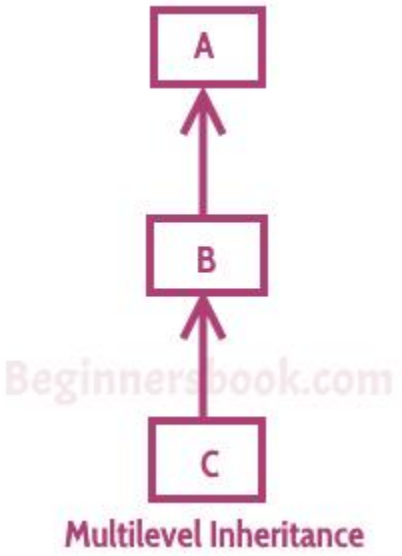
- **Constructor :** In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory. It is a special type of method which is used to initialize the object. Every time an object is created using the new() keyword, at least one constructor is called. We can pass our values , when calling out the constructor using new operator, to initialize the object's values. If there is no constructor available in the class It calls a default constructor. In such case, Java compiler provides a default constructor by default.

- **Constructor Overloading :** Java Constructor overloading is a technique in which a class can have any number of constructors that differ in parameter list. The compiler differentiates these constructors by taking into account the number of parameters in the list and their type. The constructor overloading can be defined as the concept of having more than one constructor with different parameters so that every constructor can perform a different task.

**Multilevel inheritance in java with example**

When a class extends a class, which extends anther class then this is called **multilevel inheritance**. For example class C extends class B and class B extends class A then this type of inheritance is known as multilevel inheritance.

Lets see this in a diagram:



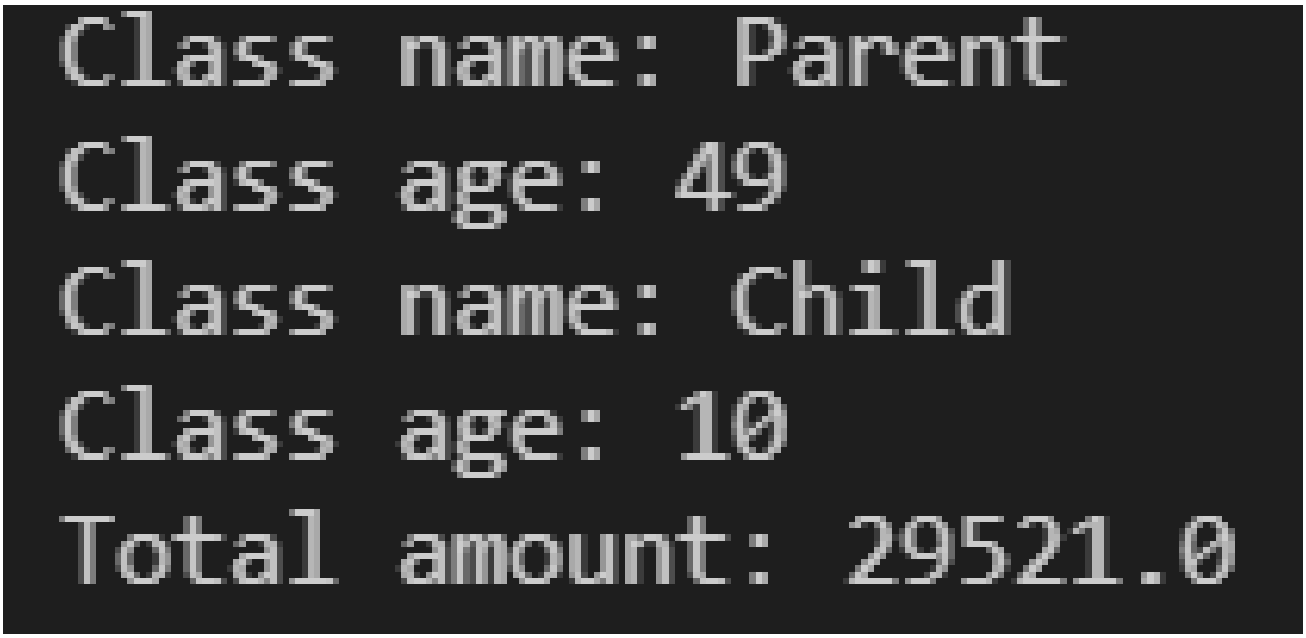

It's pretty clear with the diagram that in Multilevel inheritance there is a concept of grand parent class. If we take the example of this diagram, then class C inherits class B and class B inherits class A which means B is a parent class of C and A is a parent class of B. So in this case class C is implicitly inheriting the properties and methods of class A along with class B that's what is called multilevel inheritance.

## Source Code:

```
class grandParent{
    private double inheritance=24000.50;
    double inheritAmount(){
        return inheritance;
    }
}
class parent extends grandParent{
    private String str= "Parent";
    private int age=49;
    void displayParent(){
        System.out.println("Class name: "+str);
        System.out.println("Class age: "+age);
    }
    double inheritAmount(){
        return super.inheritAmount()+5500;
}
}
public class child extends parent{
    private String str;
    private int age;
    child(){
        str="undefined";
        age=-1;
    }
    child(String str,int age){
        this.str=str;
        this.age=age;
    }
    void displayChild(){
        System.out.println("Class name: "+str);
        System.out.println("Class age: "+age);
    }
    double inheritAmount(){
        return super.inheritAmount()+20.5;
    }
    public static void main(String[] args){
        child c=new child("Child",10);
        c.displayParent();
        c.displayChild();
        System.out.println("Total amount: "+c.inheritAmount());
    }
}
```

## Output:

```
Class name: Parent
Class age: 49
Class name: Child
Class age: 10
Total amount: 29521.0
PS C:\Users\Volted User\OneDrive\Desktop\practical>
```



# EXPERIMENT – 5.4

## Aim:

WAP illustrating all uses of super keywords.

## Theory:

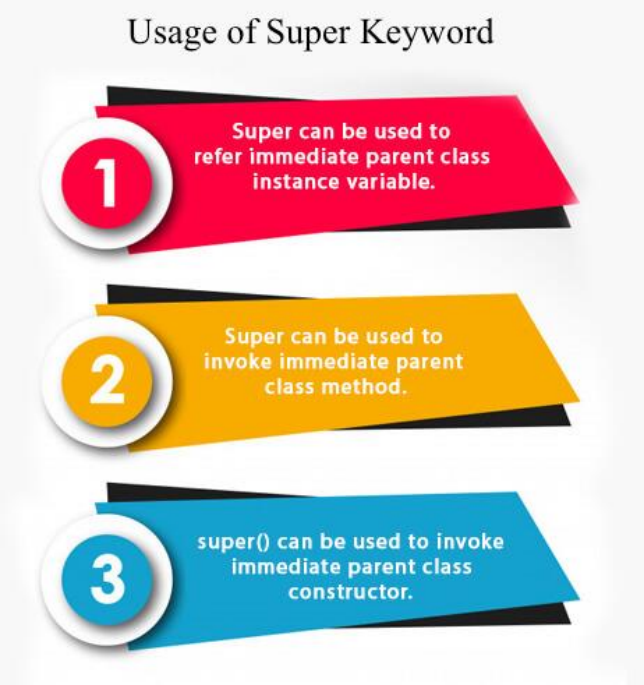
### Super Keyword in Java

The super keyword in Java is a reference variable which is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.'

### Usage of Java super Keyword

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.



1) super is used to refer immediate parent class instance variable.

2) super can be used to invoke parent class method

3) super is used to invoke parent class constructor.

## Source Code:

```
class box{
    private int a,b,c;
    box(){
        a=b=c=-1;
    }
    box(int a,int b,int c){
        this.a=a;
        this.b=b;
        this.c=c;
    }
    box(box o){
        a=o.a;
        b=o.b;
        c=o.c;
    }
    float volume(){
        return (float)a*b*c;
    }
    void shape(){
        System.out.println("It is box shape");
    }
}
class boxWeight extends box{
    private double w;
    // or we can use a for weight too, and further use super.a for superclass a
    boxWeight(){
        super();
        w=-1;
    }
    boxWeight(int a,int b,int c,double w){
        super(a,b,c);
        this.w=w;
    }
    boxWeight(boxWeight wo){
        super(wo);
```

```
        w=wo.w;
    }
    double density(){
        return w/volume();
    }
    void shape(){
        super.shape();
        System.out.println("It also have mass factor");
    }
}
public class classIntro {
    public static void main(String[] args) {
        box b1=new box();
        box b2=new box(10,15,10);
        box b3=new box(b2);
        System.out.println(b1.volume()>=0?b1.volume():"No paramteres");
        System.out.println(b2.volume()>=0?b2.volume():"No paramteres");
        System.out.println(b3.volume()>=0?b3.volume():"No paramteres");


        boxWeight b=new boxWeight(5,10,5,25.0);
        System.out.println(b.density()>=0?b.density():"No paramteres");
        b.shape();
    }

}
```

## Output:

```
No paramteres
1500.0
1500.0
0.1
It is box shape
It also have mass factor
PS C:\Users\Volted User\OneDrive\Desktop\practical>
```

# Viva Questions

## 1. When will you define a method as static?

Ans.

When a method needs to be accessed even before the creation of the object of the class then we should declare the method as static.

## 2. What are the restriction imposed on a static method or a static block of code?

Ans.

A static method should not refer to instance variables without creating an instance and cannot use "this" operator to refer the instance.

## 3. I want to print "Hello" even before main() is executed. How will you acheive that?

Ans.

Print the statement inside a static block of code. Static blocks get executed when the class gets loaded into the memory and even before the creation of an object. Hence it will be executed before the main() method. And it will be executed only once.

## 4. How-will-you-communicate-between-two-Applets

Ans.

The simplest method is to use the static variables of a shared class since there's only one instance of the class and hence only one copy of its static variables. A slightly more reliable method relies on the fact that all the applets on a given page share the same AppletContext. We obtain this applet context as follows: AppletContext ac = getAppletContext(); AppletContext provides applets with methods such as getApplet(name), getApplets(),getAudioClip, getImage, showDocument and showStatus().

## 5. Which classes can an applet extend?

Ans.

An applet can extend the java.applet.Applet class or the java.swing.JApplet class. The java.applet.Applet class extends the java.awt.Panel class and enables you to use the GUI tools in the AWT package. The java.swing.JApplet class is a subclass of java.applet.Applet that also enables you to use the Swing GUI tools.

## 6. What is the purpose of declaring a variable as final?

Ans.

A final variable's value can't be changed. final variables should be initialized before using them.

## 7. When will you define a method as static?

Ans.

When a method needs to be accessed even before the creation of the object of the class then we should declare the method as static.