

# Inheritance in java

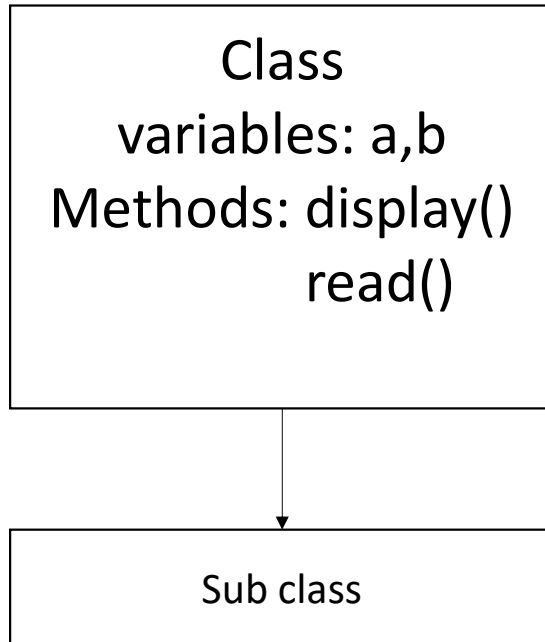
By Harshal Shah

[harshalshah43@gmail.com](mailto:harshalshah43@gmail.com)

# Inheritance

- A process by which one class can acquire properties of another class is called inheritance.
- When a class inherits from another class, it is called as the sub class.
- The class from which subclass derives its properties is called as a super class.
- When a class inherits from another class, the certain members of the parent class or super class become the part of the subclass.

## Inheritance



Based on the type of inheritance, these members get inherited i.e. become a part of the Subclass. The objects of the subclass would then Not only have subclass members but would also have the inherited parts of the super class.

# Types of inheritance

- Single
- Multilevel
- Hierarchical
- Multiple (Not supported)

# Single inheritance

- Single inheritance is when one subclass inheritance from one super class.
- The example we saw in the previous slide is an example of a single inheritance.
- It is a simplest form of inheritance.

## Program to demonstrate single inheritance

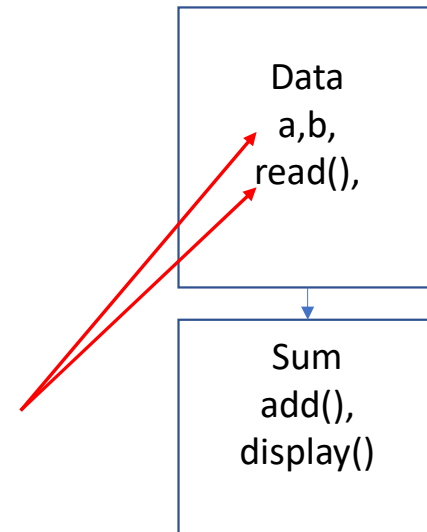
```
import java.io.*;  
import java.util.*;
```

```
class Data  
{  
protected int a, b;  
  
public void read(int x, int y)  
{  
a=x;  
b=y;  
}  
}
```

```
class Sum extends Data  
{  
private int sum;  
public void add()  
{  
sum=a+b;  
}  
public void display()  
{  
System.out.println("Sum="+sum);  
}  
}
```

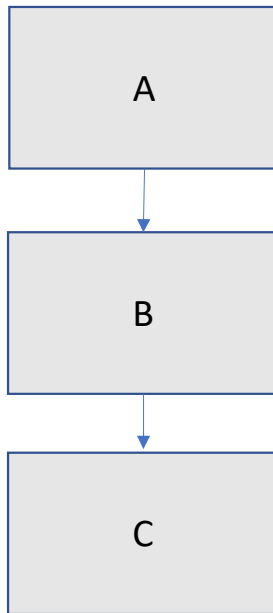
```
class singleinheritance{  
public static void main (String args[]) {  
int x,y;  
Scanner sc = new Scanner (System.in);  
System.out.println("Enter two numbers");  
x=sc.nextInt();  
y=sc.nextInt();  
Sum s=new Sum();  
s.read(x,y);  
s.add();  
s.display();  
}  
}
```

These two become the  
part of the class Sum



# Multilevel inheritance

- It is the type of inheritance where a class is inherited by a subclass, then the subclass again gets inherited by a subclass.



```

import java.util.*;
class Data{
protected float r;
public void read(float x)
{
    r=x;
}
}
class Area extends Data
{

```

```

protected float area;
public void calculate()
{
    area=3.14f*r*r;
}
public void display()
{
    System.out.println("Area="+area);
}
}

```

```

class Volume extends Area{
private float volume;
public void compute()
{
    volume=area*r*4/3;
}
public void output()
{
    System.out.println("Volume="+volume);
}
}

```

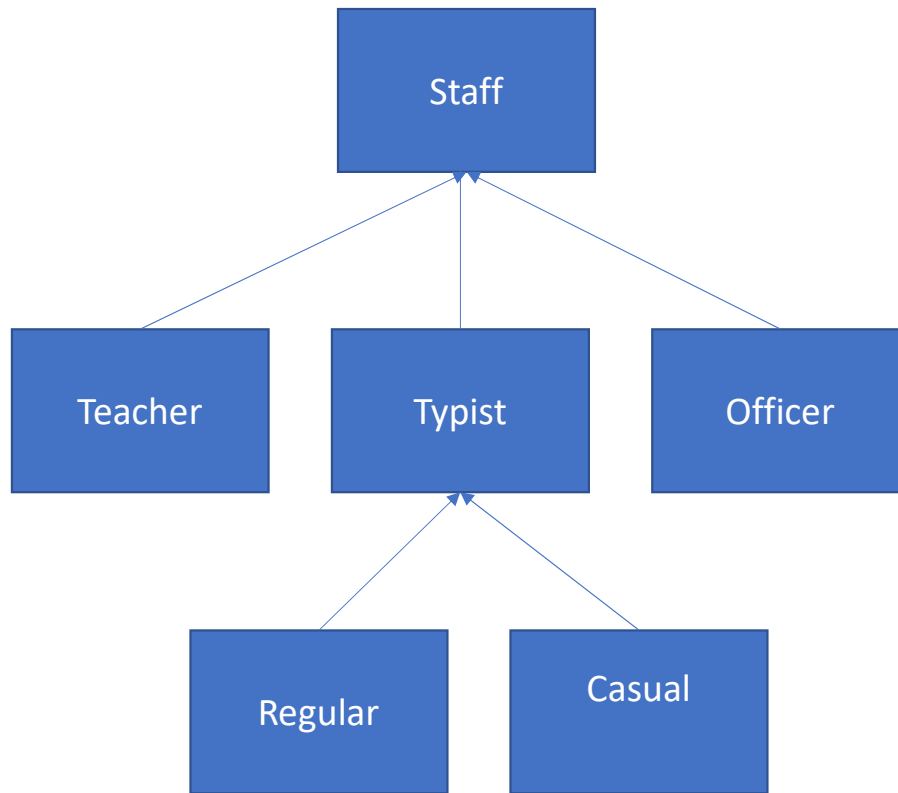
```

class Main{
public static void main (String args[]) {
    float x;
    Scanner sc = new Scanner (System.in);
    System.out.println("Enter the radius:");
    x=sc.nextFloat();
    Volume a=new Volume();
    a.read(x);
    a.calculate();
    a.display();
    a.compute();
    a.output();
}
}

```



# Hierarchical inheritance



# Method overriding

- If a class has multiple methods with the same name and different parameter list then it is called method overloading.
- If a base class and derived class have a method with the same name but same parameters, then its called method overriding.
- Example 7.6.1

# Method overriding example

Code Example:-

```
import java.util.*;
import java.io.*;
class A{
    int a= 10;
    void read(){
        System.out.println("read base");
    }
}
class B extends A{
    int a =20;
    void read()
    {
        System.out.println("read derived");
    }
}
class test{
    public static void main(String[] args) {
        B b = new B();
        b.read(); // read of derived class overrides the
read of base
    }
}
```

Explanation:-

When the method of a base class  
Has same name and same arguments  
Of a derived class, the method of  
The derived class is always called.  
This is called method overriding.

# Final keyword

- We can use this “final” keyword with any member of the class or class itself.
- If a field member is declared as final then the variable value cannot be changed i.e it becomes a constant.
- If a method is declared final, it cannot be overridden.
- If a class is declared as final then that class cannot have any subclass.

## Let us see an example of a final method

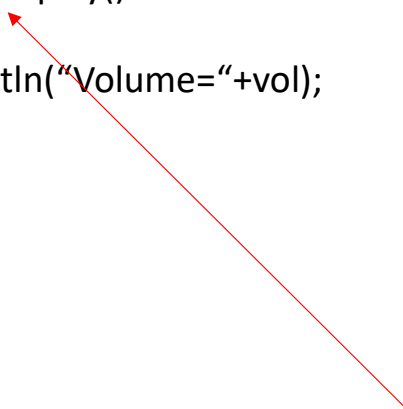
```
import java.util.*;
```

```
class Base{  
    protected float r,vol;  
    public void read(float x)  
    {  
        r=x;  
    }  
    final public void display()  
    {  
        System.out.println("Volume="+vol);  
    }  
}
```

```
class Sphere extends Base  
{  
    public void calculate()  
    {  
        vol=3.14f*r*r*r*r*4/3;  
    }  
}
```

```
class Hemisphere extends Base  
{  
    public void calculate()  
    {  
        vol=3.14f*r*r*r*r*2/3;  
    }  
}
```

```
class Main{  
    public static void main (String args[]) {  
        float x;  
        Scanner sc = new Scanner (System.in);  
        System.out.println("Enter the radius:");  
        x=sc.nextFloat();  
        Sphere s=new Sphere();  
        s.read(x);  
        s.calculate();  
        System.out.println("Sphere:");  
        s.display();  
        Hemisphere h=new Hemisphere();  
        h.read(x);  
        h.calculate();  
        System.out.println("Hemisphere:");  
        h.display();  
    }  
}
```



Notice how the display() method is declared final.

```
import java.util.*;
class Staff
{
    protected String name;
    protected int code;
}
class Teacher extends Staff
{
    private String subject;
    private int experience;
    public void read()
    {
        Scanner sc = new Scanner (System.in);
        System.out.println("Enter name, code, subject and experience of the teacher:");
        name=sc.next();
        code=sc.nextInt();
        subject=sc.next();
        experience=sc.nextInt();
    }
    public void display()
    {
        System.out.println("Teacher Details:\nName:"+name+"\nCode:"+code
        +"\nSubject:"+subject+"\nExperience:"+experience);
    }
}
```

```
class Officer extends Staff
{
    private    String dept;
    private int grade;
    public    void read()
    {
        Scanner sc= new Scanner(System.in);
        System.out.println("Enter name, code, department and grade of the officer:");
        name =sc.next();
        code =sc.nextInt();
        dept =sc.next();
        grade =sc.nextInt();
    }
    public void display()
    {
        System.out.println("Officer Details:\nName:"+name+"\nCode:"+code+
            "\nDepartment:"+dept+"\nGrade:"+grade);
    }
}
class Typist extends Staff
{
    protected int speed,experience;

}
```

```

class Regular extends Typist
{
    private    float salary;
    public void read()
    {
        Scanner sc = new Scanner (System.in);
        System.out.println("Enter name, code, speed,
experience and salary of the          regular typist:");
        name=sc.next();
        code=sc.nextInt();
        speed=sc.nextInt();
        experience=sc.nextInt();
        salary=sc.nextFloat();
    }
    public void display()
    {
        System.out.println("Regular Typist
Details:\nName:"+name+"\nCode:"+code+"\nSpeed:"+speed+"\nExperience:"+e
xperience+"\nSalary:"+salary);
    }
}

```



```

class Casual extends Typist
{
    private float dailywages;
    public void read()
    {
        Scanner sc = new Scanner (System.in);
        System.out.println("Enter name, code, speed,
experience and daily wages of           the Casual typist:");
        name=sc.next();
        code=sc.nextInt();
        speed=sc.nextInt();
        experience=sc.nextInt();
        dailywages=sc.nextFloat();
    }
    public void display()
    {
        System.out.println("Casual Typist
Details:\nName:"+name+"\nCode:"+code+"
        \nSpeed:"+speed+" \nExperience:"+experience+"\nDaily
Wages:"+dailywages);
    }
}

```

# Abstract class and method

- Abstract class are used to declare common characteristics of subclasses.
- Abstract classes are declared using the keyword “abstract” preceding the class definition. Abstract classes are used to provide a template for subclasses.
- No object can be made of an abstract class. It can be used as a base class for other classes that are derived from the abstract class.
- An abstract class contain fields and methods.
- Refer 7.8.1

# Constructors in java

- Constructors cannot be inherited.
- However the derived classes can access the base class constructors using the super keyword.
- Let us see an example below

# Super keyword

- If you want to access the base class member from the derived class, then we use the super keyword.
- This is especially to access the constructors and method members of the super class.
- Generally it is used for accessing those members of the base class that are not inherited or cannot be inherited.
- We know that the constructors cannot be inherited.
- So if a base class has a constructor and the derived class object needs to call the base constructor, then we can create a derived class constructor and then within its definition we can use the super keyword to call the base class constructor.

# When to use super keyword

- When an object of a subclass is created, an instance of a [aren't class is implicitly created that can be referenced using super keyword.
  - A super keyword is used to access the base class variable
  - A super keyword is used to access the base class method
  - A super keyword is used to access the base class constructor

# 1. Super to access base class variable

```
6  import java.io.*;
7
8  class Animal
9  {
10     protected String color="white";
11 }
12
13 class Dog extends Animal
14 {
15     String color="black";
16     void printColor()
17     {
18         System.out.println(color);//prints color of Dog class
19         System.out.println(super.color);//prints color of Animal class
20     }
21 }
22
23 class super1
24 {
25     public static void main(String[] args) {
26         Dog d = new Dog();
27         d.printColor();
28     }
29 }
```

## 2. Super to access class method()

```
1 //class method - super.method()
2
3 import java.io.*;
4 class Animal{
5     void eat(){
6         System.out.println("eating...");
7     }
8 }
9 class Dog extends Animal
10 {
11     void eat(){
12         System.out.println("eating bread...");
13     }
14     void bark(){
15         System.out.println("barking...");
16     }
17     void work(){
18         super.eat();
19         bark();
20         eat();
21     }
22 }
23 class super2{
24     public static void main(String[] args) {
25         Dog d = new Dog();
26         d.work();
27     }
```

### 3. Super to access base class constructor

```
1 //constructors - super()
2
3 import java.io.*;
4
5 class Animal{
6     int a=100;
7     Animal(String a)
8     {
9         System.out.println("animal is created"+a);
10    }
11
12 }
13 class Dog extends Animal{
14     int a=200;
15     Dog(String x,String y){
16         super(x);
17         System.out.println("dog is created"+x+y);
18     }
19     Animal animal = new Animal("ABC");
20 }
21 class super3
22 {
23     public static void main(String[] args) {
24         Dog d = new Dog("A","B");
25     }
26 }
27 }
```



# Explanation

- An object `c` is created of the child class. The constructor as we know is automatically called.
- In the child class, we have the constructor called `child` which can call the constructor of the parent class using `super()`.
- When `super` keyword is written, without specifying the method name, the constructor of the base class is called.
- The `display()` method is then called for the child class. Again using the keyword “`super`”, the `display()` method of the parent class is called.
- This is how we can prevent method overriding.

# Interface

- Java doesn't support multiple inheritance.
- This problem is solved in java using Interface.
- Some features of interface:
  - One interface can extend as many interfaces as required.
  - A java class can implement as many interfaces as required.
  - A class that implements an interface has to either define all methods of that interface or declare abstract methods whose definitions may be defined in the subclass.

# Extending the interface

- Interface can be used for defining a general template and then classes can implement the interface and hence inherit the properties of the “interface”.
- Interface just specify the method declaration and can also contain fields.
- Methods of the interface must be public or abstract.

# Variables in the interface

- The fields are implicitly public static final.
- The definition of the interface begins with the keyword interface.
- No object can be made of an interface i.e. it cannot be instantiated.
- Refer program 7.9.1