Inheritance

Course: CSL304

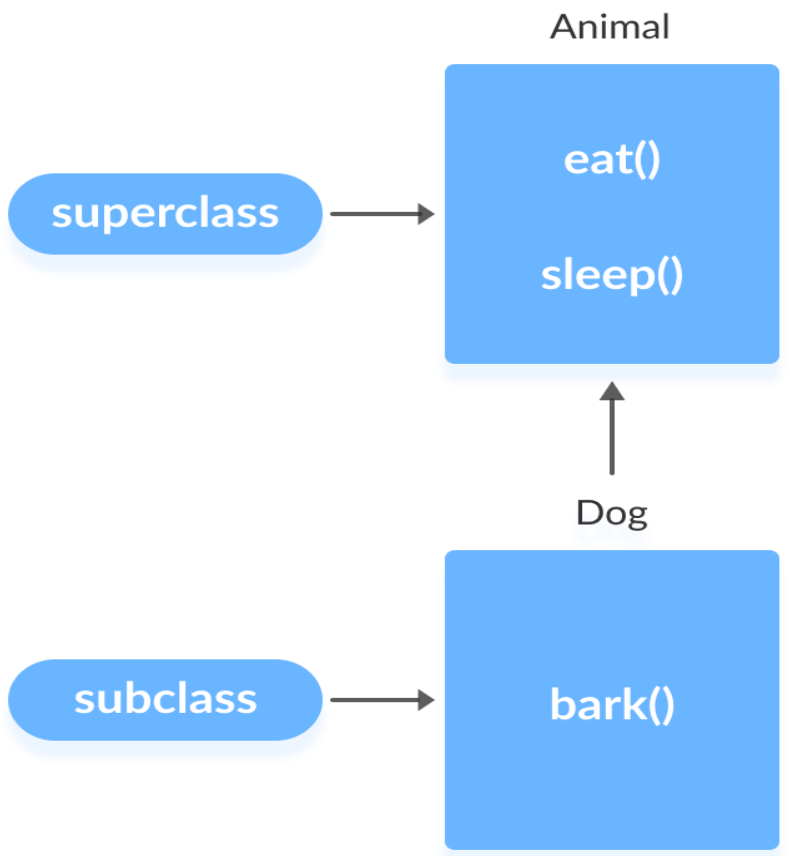
Prof. Juhi Ganwani

CONTENT

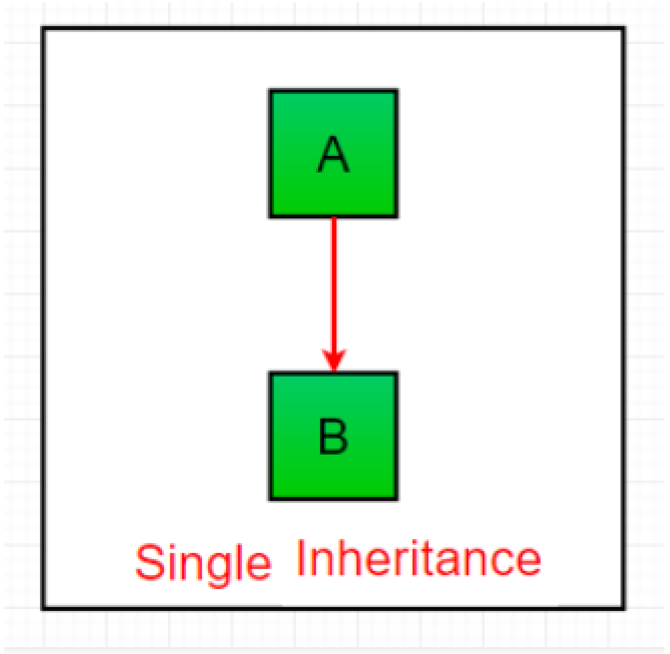
Inheritance

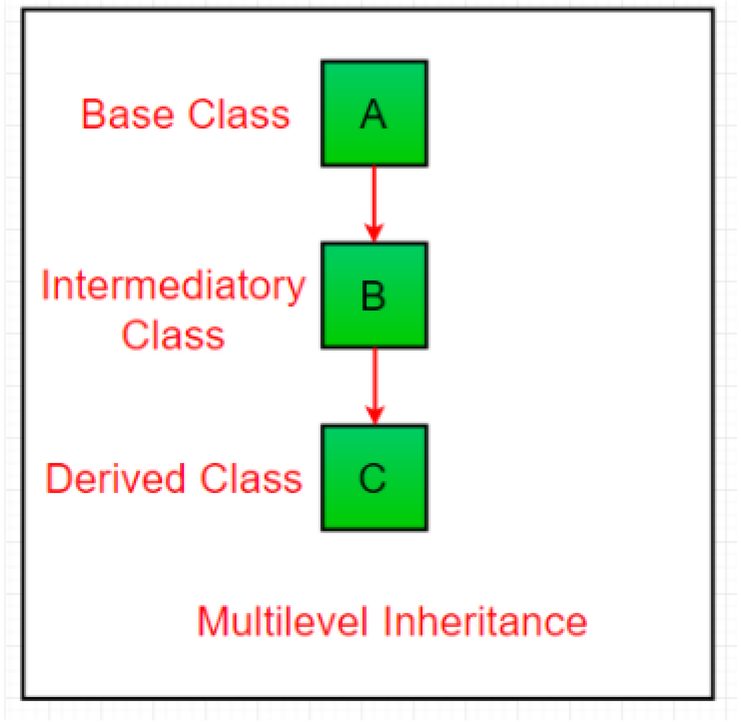
- Concept of reusability
- Mechanism of deriving a new class from an old one
- New class derive properties from base class
- Old class: base class or super class or parent class
- New class: derived class or sub class or child class
- To inherit from a class, use the extends keyword.

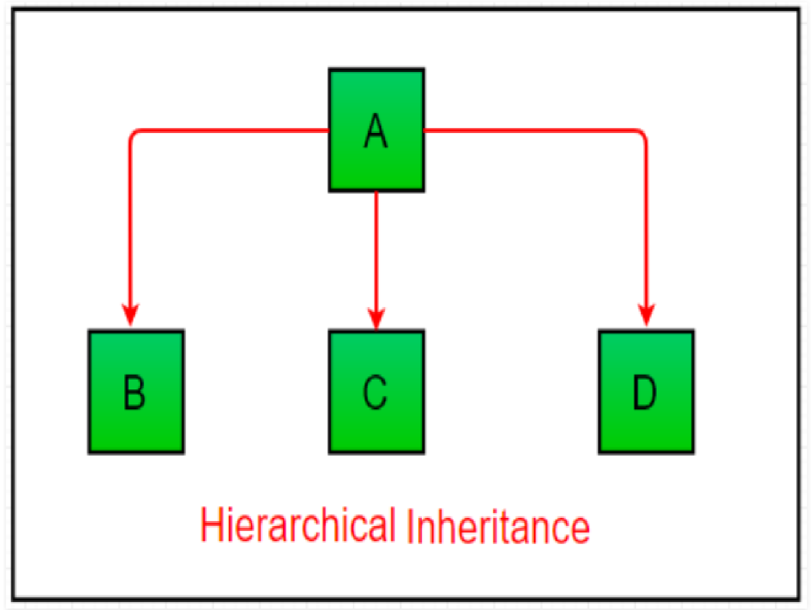
```
class derived-class extends base-class
{
  //methods and fields
}
```



Types of Inheritance

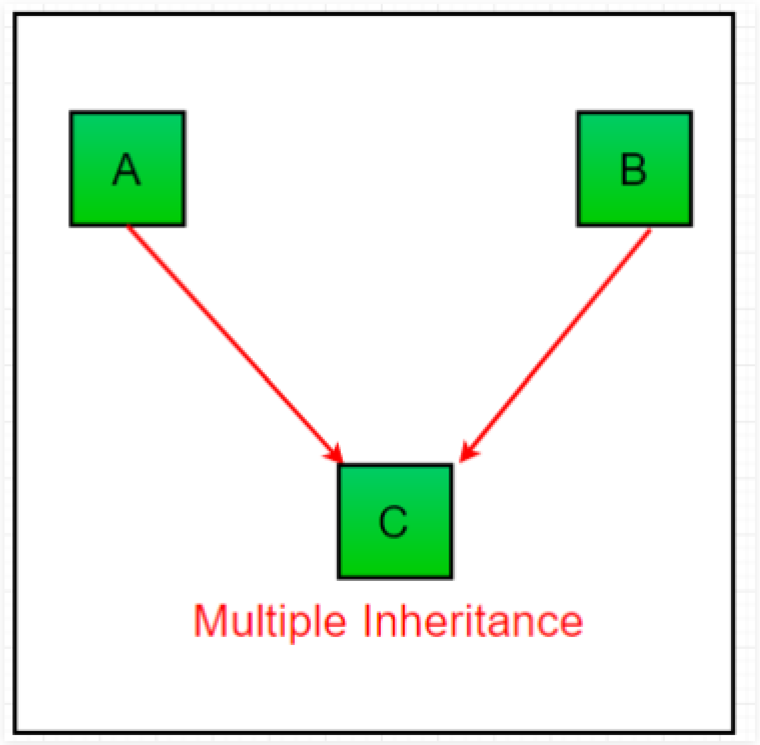






Multiple inheritance (through interfaces)

- Java does not support multiple inheritance using classes
- Class C is inherited from interface A and B



Single Inheritance

```
class Rectangle
{
int length, breadth;
Rectangle(int x,int y)
{length=x;              breadth=y;}
int area(){
  return length*breadth;}
}
class RectangularPrism extends Rectangle
{
int height;
RectangularPrism(int x,int y,int z){
super(x,y);
height=z;}
int volume(){
return length*breadth*height;}}
```

```
class SingleInheritance
{
public static void main(String args[])
{
        RectangularPrism prism=new RectangularPrism(2,3,10);
        int area1=prism.area();
        int vol1=prism.volume();
        System.out.println("Area="+area1);
        System.out.println("Volume="+vol1);
}
}
```

Note:

Subclass constructor uses the keyword super to invoke the constructor method of the superclass

Call to superclass constructor must appear as the first statement within the subclass constructor

super keyword

- It is similar to this keyword
- It is used to differentiate members of superclass from subclass, if they have same names
- It is used to invoke superclass constructor from subclass

```
class Super_class
{
int num = 20;
public void display()
{
System.out.println("This is the display method
of superclass");
}
}

public class Sub_class extends Super_class
{
int num = 10;
public void display()
{System.out.println("This is the display method
of subclass");}
```

```
public void my_method()
{
Sub_class sub = new Sub_class();
sub.display();
super.display();
System.out.println("value of the variable named num
in sub class:"+ sub.num);
System.out.println("value of the variable named num
in super class:"+ super.num);
}

public static void main(String args[])
{
        Sub_class obj = new Sub_class();
        obj.my_method();
 }
}
```

Multilevel Inheritance

```
class Shape {
  public void display() {
    System.out.println("Inside display");
  }
}
class Rectangle extends Shape {
  public void area() {
    System.out.println("Inside area");
  }
}
```

```
class Cube extends Rectangle {
  public void volume() {
    System.out.println("Inside volume");
  }
}
public class CubeMain {
  public static void main(String[] arguments) {
    Cube cube = new Cube();
    cube.display();
    cube.area();
    cube.volume();
  }
}
```

Hierarchical Inheritance

```
class Animal{
void eat()
{System.out.println("eating...");}
}
class Dog extends Animal{
void bark()
{System.out.println("barking...");}
}
class Cat extends Animal{
void meow()
{System.out.println("meowing...");}
}
Class AnimalMain{
public static void main(String args[]){
Cat c=new Cat();
c.meow();
c.eat();
//c.bark();        //C.T.Error
}}
```

Why Multiple Inheritance is not allowed?

- To reduce the complexity and simplify the language
- Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes.
- If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

Example

```
class A{
void msg(){System.out.println("Hello");}
}
class B{
void msg(){System.out.println("Welcome");}
}
class C extends A,B{//suppose if it were

 public static void main(String args[]){
   C obj=new C();
   obj.msg();   //Now which msg() method would be invoked?
}
}
```

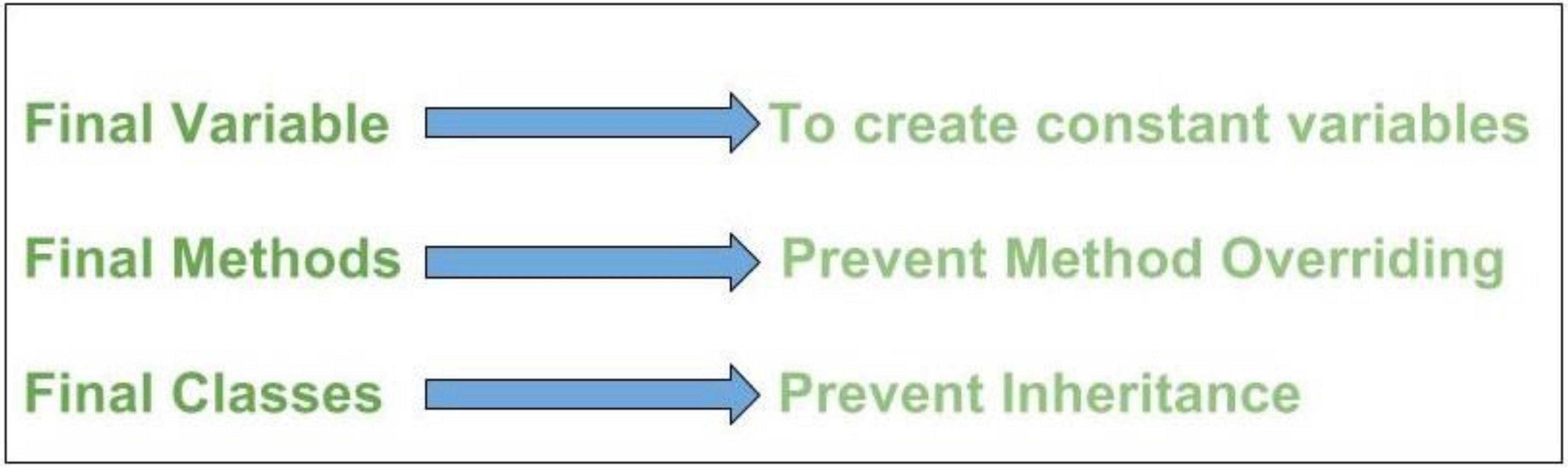
Method overriding

- Methods having same name & argument list of which one is in superclass & other is in subclass.
- Child class can give its own implementation to a method which is already provided by the parent class.
- Method in parent class is called overridden method and the method in child class is called overriding method.
- If a call is made from subclass, then it will refer to subclass method thereby hiding the method of the superclass.
- If a method call is outside the superclass & subclass then the method invoked will be based object invoking it.

```
class Super
{
void display(){
System.out.println("Super class");
}
}
class Sub extends Super
{
void display(){
System.out.println("Sub class");
super.display();
}
}
```

```
class OverrideTest
{
public static void main(String args[])
	{
		Super s1=new Super();
		s1.display();
		Sub s2=new Sub();
		s2.display();
	}
}
```

final keyword



final variable

- When a variable is declared with *final* keyword, its value cannot be modified.

```
class Bike
{
        final int speedlimit=90;
        void run(){
                speedlimit=400;
        }
        public static void main(String args[]){
                Bike obj=new Bike();
                obj.run();
        }
}
```

final method

- We cannot override any final method

```
class Bike{
        final void run(){
                System.out.println("running");}
}
class Honda extends Bike
{
        void run(){
                System.out.println("Running safely with 100kmph");}

        public static void main(String args[]){
                Honda h=new Honda();
                h.run();
        }
}
```

final class

- Final class cannot be extended

```
final class Bike
{}
class HondaClass extends Bike
{
        void run(){
                System.out.println("Running safely with 100kmph");}
        public static void main(String args[])
        {
                HondaClass h=new HondaClass();
                h.run();
        }
}
```

Abstraction

- Data **abstraction** is the process of hiding certain details and showing only essential information to the user.
- **Abstraction** - hiding the implementation details and showing only functionality to the user.
- Example: email, sms, etc
- It can be achieved using abstract classes or interfaces
- **Abstract class:** is a restricted class that cannot be used to create objects
- **Abstract method:** can only be used in an abstract class, and it does not have a body. The body is provided by the subclass

Rules of abstract class

- It may or may not contain *abstract methods*, i.e., methods without body
- If a class has at least one abstract method, then the class **must** be declared abstract.
- To use an abstract class, you have to inherit it from another class, provide implementations to the abstract methods in it.
- If you inherit an abstract class, you have to provide implementations to all the abstract methods in it.

Abstract class

```
abstract class Animal {
  public abstract void animalSound();
  public void sleep() {
    System.out.println("Zzz");
  }
}
```

Example

```
abstract class Animal {
  public abstract void animalSound();
  public void sleep() {
    System.out.println("Zzz");}
}
class Dog extends Animal {
  public void animalSound() {
    // The body of animalSound() is provided here
    System.out.println("The Dog says: woof woof");
  }
}
```

```
class MyMainClass {
  public static void main(String[] args) {
    Dog mydog = new Dog();
    mydog.animalSound();
    mydog.sleep();
  }
}
```

Interface

- It is similar to a class
- Methods declared in an interface is by default abstract and variables are public, static & final
- Interfaces define only abstract methods & final fields.
- Class has to implement interface to define the code for implementation of these methods

Syntax:

```
interface InterfaceName
{
        variables declaration;
        methods declaration;
}
```

Interface

Example:

```
interface Area
{
        final static float pi=3.142;
        float compute(float x, float y);
        void show();
}
```

Implementing interface

- Class can implement more than one interface
- Interface can extends another interfaces
- A class that implements interface must implement all the methods in interface otherwise that class should be declared as abstract class

Example

```
interface Area
     {
        final static double pi = 3.14;
        double calc(double x,double y);
      }

class Rectangle implements Area
  {
     public double calc(double x,double y)
      {
         return(x*y);
       }
  }
```

```
class Circle implements Area
  {    public double calc(double x,double y)    {
           return(pi*x*y);
         }
  }

class InterfaceTest
 {
    public static void main(String arg[])    {
        Rectangle r = new Rectangle();
        Circle c = new Circle();
        System.out.println("\nArea of Rectangle is : " +r.calc(10,20));
        System.out.println("\nArea of Circle is : " +c.calc(15,15));     }
 }
```

Partial implementation of an interface

```
interface Area
{
        double pi=3.14;
        double compute(double x,double y);
}
abstract class Circle implements Area
{
        public void display()
        {
                System.out.println(“Circle Class”);
        }
}
```

Extending interfaces (multiple inheritance)

```
interface One {
   public void methodOne();
}
interface Two {
   public void methodTwo();
}
class Three implements One, Two {
   public void methodOne()
   { }
   public void methodTwo()
   {}
}
```

Note: A class can extend a class and can implement any number of interfaces simultaneously.

Interface Examples

- Write a class Rectangle which implements interface Polygon having method getArea with parameters length and breadth. (Rectangle Main)
- Write a class Bird which implements two interfaces BirdEat & BirdFly having method eating & flying respectively. (BirdMain)
- Write a class College which inherits class Name and implements two interfaces Teacher & Student having method teach and study respectively. (CollegeTest.java)