Java Inheritance

Introduction

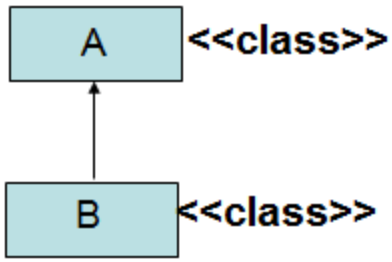
1. Reusability is achieved by INHERITANCE
2. Java classes Can be Reused by extending a class. Extending an existing class is nothing but reusing properties of the existing classes.
3. The class whose properties are extended is known as super or base or parent class.
4. The class which extends the properties of super class is known as sub/ derived / child class
5. A class can either extends another class or can implement an interface

Introduction(contd..)

- Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship.
- **Why use inheritance in java**
  - For Code Reusability
  - For Method Overriding
  - (so runtime polymorphism can be achieved)

Syntax:

class B extends A { ….. }



Syntax :

class <subclass name> *extends* <superclass name>

{

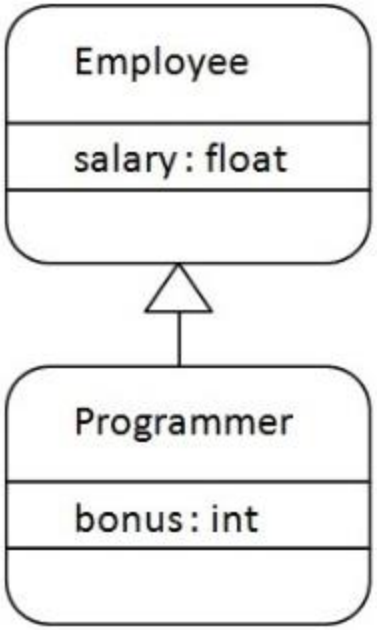
variable declarations;

method declarations;

}

- **Extends keyword signifies that properties of the super class are extended to sub class**
- **Sub class will not inherit private members of super class**

Example: Inheritance




```
class Employee{
 float salary=40000;
}

class Programmer extends Employee
{
 int bonus=10000;
 public static void main(String args[])
{
 Programmer p=new Programmer();
 System.out.println("Programmer salary is:”
+p.salary);
 System.out.println("Bonus of Programmer is:"+p.bonus);
 }
}
```

Output:

Programmer salary is: 40000

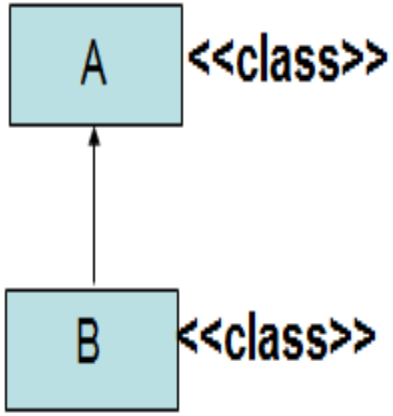
Bonus of Programmer is: 10000

Types of Inheritance

1. **Single**
2. **Multilevel**
3. **Hierarchical**
4. **Multiple**
5. **Hybrid**

1. Single Inheritance

- Inheritance in which a class extends another **one** class **only** then we call it a **single inheritance.**
- Diagram shows that class B extends only one class which is A.
- Here A is a **parent class** of B and B would be a **child class** of A.



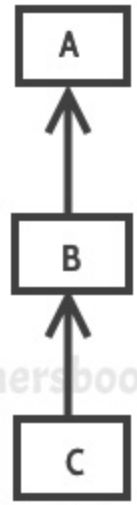
Example: Single Inheritance

```
Class A
{
  public void methodA()
  {
    System.out.println("Base class method");
  }}
Class B extends A
{
  public void methodB()
  {
    System.out.println("Child class method");
  }
  public static void main(String args[])
  {
    B obj = new B();
    obj.methodA(); //calling super class method
    obj.methodB(); //calling local method
  }
```

Output:
Base class method
Child class method

2. Multilevel Inheritance

- **Multilevel inheritance** refers to a mechanism in OO technology where one can inherit from a derived class, thereby making this derived class the base class for the new class.
- As you can see in below flow diagram C is subclass / child class of B and B is a child class of A.



Multilevel Inheritance

Example: Multilevel Inheritance

```
Class X
{
  public void methodX()
  {
    System.out.println("Class X method");
  }



}
Class Y extends X
{
public void methodY()
{
System.out.println("class Y method");
}
}
```

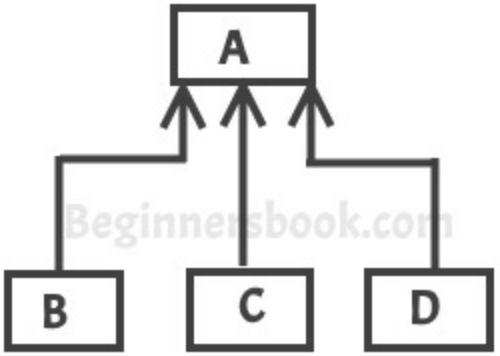
```
Class Z extends Y
{
  public void methodZ()
  {
    System.out.println("class Z method");
  }

  public static void main(String args[])
  {
    Z obj = new Z();
    obj.methodX(); //grand parent class method
    obj.methodY(); //calling parent class method
    obj.methodZ(); //calling local method
  }
}
```

Output:

Class X method
class Y method
class Z method

3. Hierarchical Inheritance

- When more than one classes inherit a same class then this is called hierarchical inheritance.
- For example class B, C and D extends a same class A.



Hierarchical Inheritance

Example: Hierarchical Inheritance

```
class A
{
  public void methodA()
  {
    System.out.println("method of Class A");
  }
}
class B extends A
{
  public void methodB()
  {
    System.out.println("method of Class B");
  }
}
class C extends A
{
  public void methodC()
  {
    System.out.println("method of Class C");
```

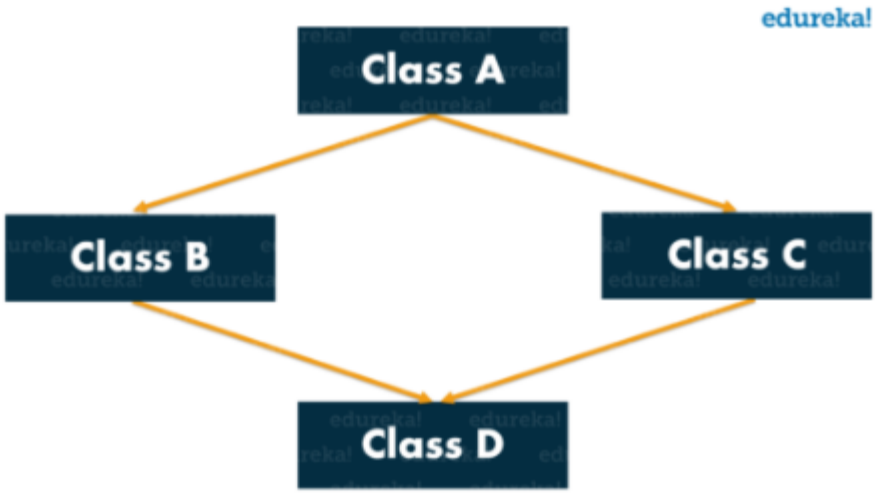
```
class D extends A
{
  public void methodD()
  {
    System.out.println("method of Class D");
  }
}

class JavaExample
{
  public static void main(String args[])
  {
    B obj1 = new B();
    C obj2 = new C();
    D obj3 = new D();

    //All classes can access the method of class A
    obj1.methodA();
    obj2.methodA();
    obj3.methodA();
  }
}
```

Output:

```
method of Class A
method of Class A
method of Class A
```

4. Multiple Inheritance

- Multiple inheritance refers to the process where one child class tries to extend more than one parent class



NOT SUPPORTED IN JAVA…WHY???

4. Multiple Inheritance (contd..)

```
// First Parent class
class Parent1
{
   void fun()
   {
      System.out.println("Parent1");
   }
}

// Second Parent Class
class Parent2
{
   void fun()
   {
      System.out.println("Parent2");
   }
}
```

```
// Error : Test is inheriting from
//multiple  classes
class Test extends Parent1, Parent2
{
   public static void main(String args[])
   {
      Test t = new Test();
      t.fun();
   }
}
```

Output:

Compile Error

4. Multiple Inheritance (co

1. The Diamond Problem:

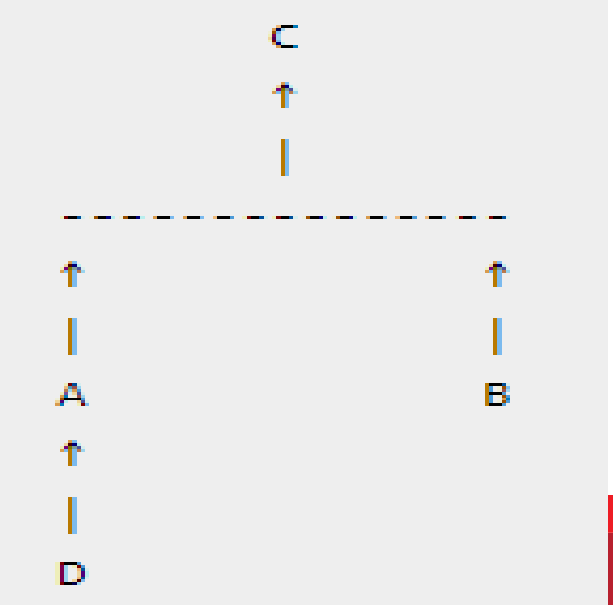
```
        GrandParent
         /     \
        /       \
    Parent1     Parent2
        \       /
         \     /
          Test
```

```
// A Grand parent class in diamond
class GrandParent
{
   void fun()
   {
      System.out.println("Grandparent");
   }
}

// First Parent class
class Parent1 extends GrandParent
{
   void fun()
   {
      System.out.println("Parent1");
   }
}
```

```
// Second Parent Class
class Parent2 extends GrandParent
{
   void fun()
   {
      System.out.println("Parent2");
   }
}

// Error : Test is inheriting from multiple
// classes
class Test extends Parent1, Parent2
{
   public static void main(String args[])
   {
      Test t = new Test();
      t.fun();
   }
}
```

5. Hybrid Inheritance

- A hybrid inheritance is a combination of more than one **types of inheritance**
- For example when class A and B extends class C & another class D extends class A then this is a hybrid inheritance, because it is a combination of single and hierarchical inheritance

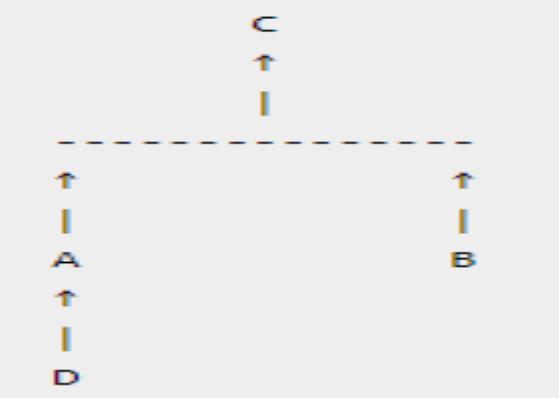


Example: Hybrid Inheritance

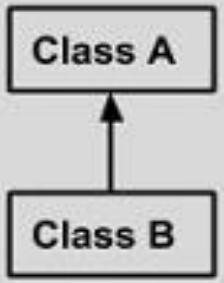
```
class C
{
  public void disp()
  {
            System.out.println("C");
  }
}

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

class B extends C
{
  public void disp()
  {
            System.out.println("B");
  }

}
```

```
class D extends A
{
  public void disp()
  {
            System.out.println("D");
  }
  public static void main(String args[]){

            D obj = new D();
            obj.disp();
  }
}
```

Output:
D

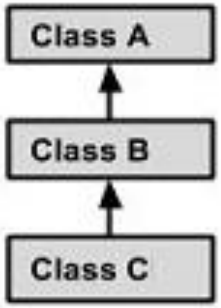


Single Inheritance



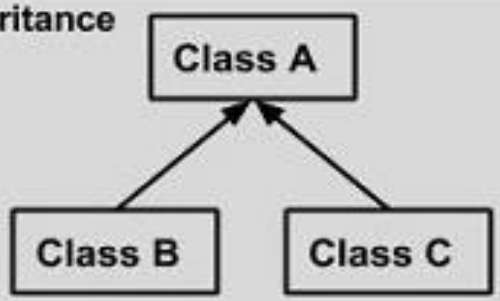

```
public class A {
   .......
}
public class B extends A {
   .........
}
```

Multi Level Inheritance



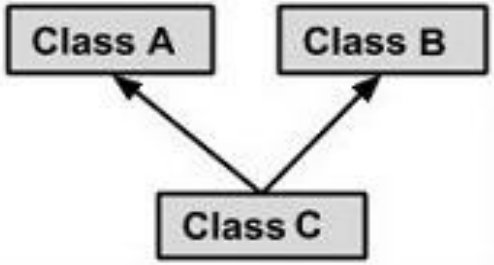

```
public class A { ...................}

public class B extends A {...................}

public class C extends  B {..................... }
```

Hierarchical Inheritance




```
public class A { ...................}

public class B extends A {...................}

public class C extends A {..................... }
```

Multiple Inheritance




```
public class A { ...................}

public class B {...................}

public class C extends  A,B {
   ....................
} // Java does not support mutiple Inheritance
```

Super keyword in Java

- The super keyword refers to the objects of immediate parent class.
- **The use of super keyword**

1. **To refer immediate parent class instance variable.**

- It is used if parent class and child class have same fields.

2. **To invoke parent class constructor.**

- It is used to invoke the parent class constructor

Super keyword: To access parent class variable

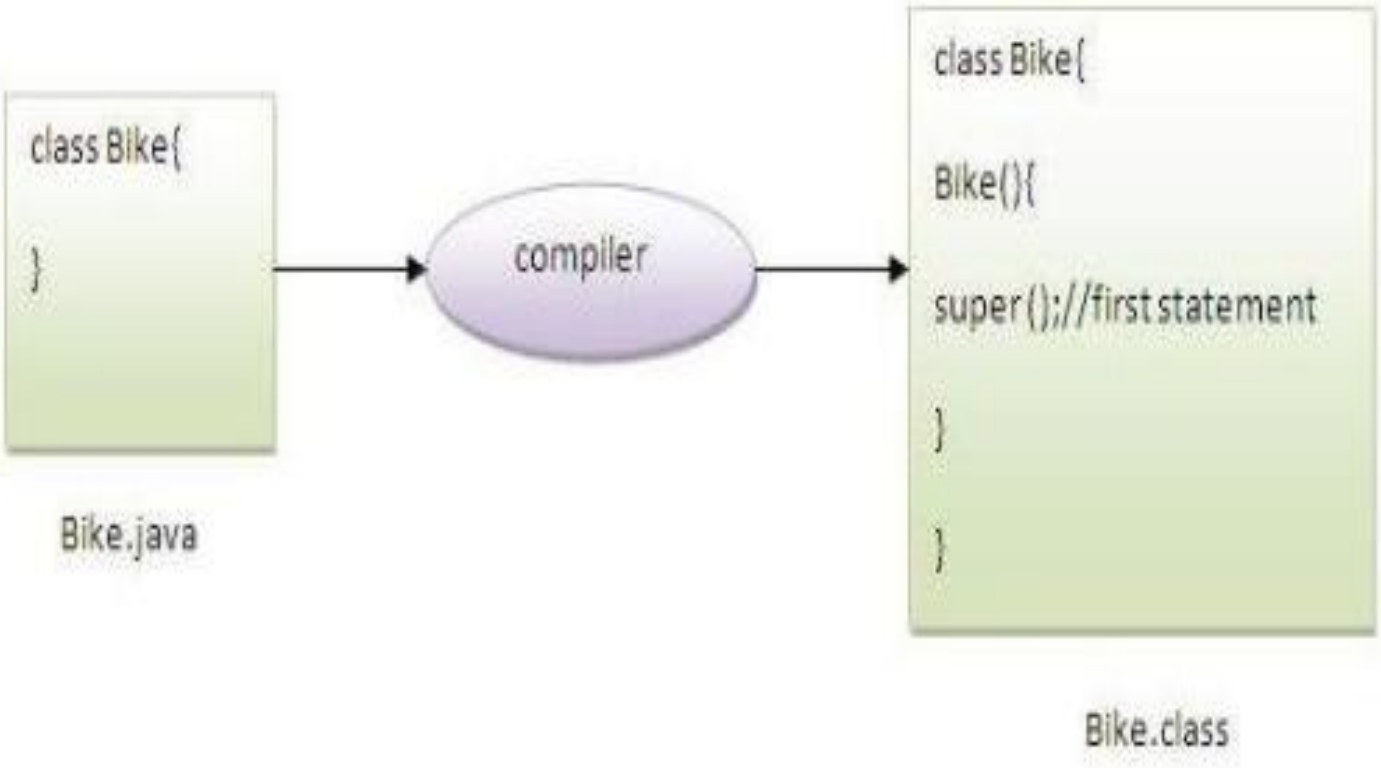
```
class Superclass
{
   int num = 100;
}

class Subclass extends Superclass
{
int num = 110;
void printNumber()
{
            System.out.println(num);
 }

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

Output:
110

```
class Superclass
{
   int num = 100;
}

class Subclass extends Superclass
{
   int num = 110;
   void printNumber()
{
            System.out.println(super.num);
   }
   public static void main(String args[]){
            Subclass obj= new Subclass();
            obj.printNumber();
   }
}
```

Output:
100

super keyword: invoke constructor of parent class



super keyword: invoke constructor of parent class

```
class Parentclass
{
  Parentclass()
{
System.out.println("Constructor of parent class");
  }
}
class Subclass extends Parentclass
{
Subclass(){
/* Compiler implicitly adds super() here as the first statement of this constructor. */
System.out.println("Constructor of child class");
  }
 Subclass(int num) {
System.out.println("arg constructor of child class");
}
 void display()
{
System.out.println("Hello!");
}

public static void main(String args[]){

        Subclass obj= new Subclass();
        obj.display();

        Subclass obj2= new Subclass(10);
        obj2.display();

  }
}
```

Output:

Constructor of parent class
Constructor of child class
Hello!
Constructor of parent class
arg constructor of child class
Hello!

Parameterized super() call: invoke parameterized constructor of parent class

```
class Parentclass
{
Parentclass()
{
System.out.println("no-arg constructor of parent class");
  }
Parentclass(String str)
{
System.out.println("parameterized constructor of parent class");
   }
}
```

```
class Subclass extends Parentclass
{
  Subclass()
{
/* super() must be added to the first statement of constructor  otherwise you will get a compilation error. */

super("Hahaha");
System.out.println("Constructor of child class");
 }
  void display()
{ System.out.println("Hello");  }

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

Output:

parameterized constructor of parent class
Constructor of child class
Hello

Example: Parent Class constructor

```
class Person
{
int id;
String name;
Person(int id,String name)
{
this.id=id;
this.name=name;
}
}
class Emp extends Person
{
float salary;
Emp(int id,String name,float salary)
{
super(id,name);//reusing parent constructor
this.salary=salary;
}
void display()
{
System.out.println(id+" "+name+" "+salary);
}
}

class TestSuper5
{
public static void main(String[] args)
{
Emp e1=new Emp(1,"ankit",45000f);
e1.display();
}
}
```

Output:

1 ankit 45000

super keyword: invoke parent class method

```
class Animal
{
void eat()
{   System.out.println("eating...");}
}
class Dog extends Animal
{
void eat()
{
System.out.println("eating bread...");
}
void bark()
{
System.out.println("barking...");
}
void work()
{
super.eat();
bark();
}
}
class TestSuper2
{
public static void main(String args[])
{
Dog d=new Dog();
d.work();
}
}
```

Output:

eating...
barking...

Method Overriding

- Declaring a method in **sub class** which is already present in **parent class** is known as method overriding.

- Overriding is done so that a child class can give its own implementation to a method which is already provided by the parent class.

- In this case the **method in parent class is called overridden method** and the **method in child class is called overriding method.**

Example: Method overriding

```
class Human{
  //Overridden method
  public void eat()
  {
    System.out.println("Human is eating");
  }
}
class Boy extends Human{
  //Overriding method
  public void eat(){
    System.out.println("Boy is eating");
  }

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

Output:
Boy is eating

Super keyword in Method Overriding

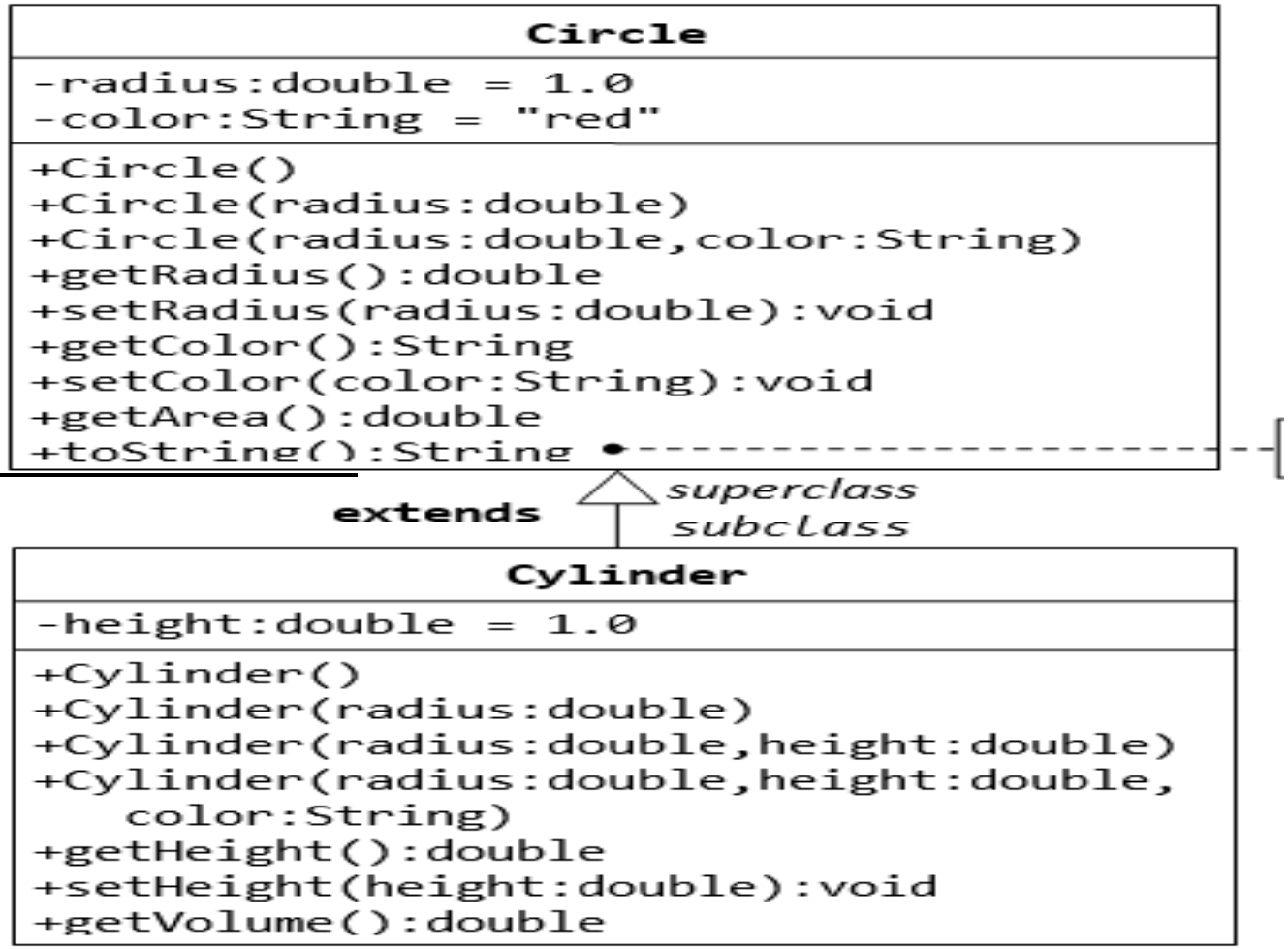
```
class Parentclass
{
  //Overridden method
  void display()
{
                System.out.println("Parent class method");
  }
}

class Subclass extends Parentclass
{
  //Overriding method
  void display(){
System.out.println("Child class method");
  }
  void printMsg(){
                display();                                  //This would call Overriding method
            super.display();                    //This would call Overridden method
  }
  public static void main(String args[]){
                Subclass obj= new Subclass();
                obj.printMsg();
  }
}
```

Output:
Child class method
Parent class method

Assignment: 1

- Create class Account which has method accountholder() to print accountholder details like account number, name, address, phone_number, balance
- Create subclass class Saving_Account which calculate_interest() based on interest rate given by user and display_balance() after deducting withdrawal amount
- Create subclass class Current_Account which calculate_interest() based on interest rate given by user and display_balance() after deducting withdrawal amount
- Create class Example which reads input from user to demonstrate inheritance concept with super keyword concept

Assignment 2:



- Write a test program (TestCylinder) to test the Cylinder class created