METHOD OVERRIDING.

NEED:

In inheritance, the parent class and child class may sometimes have same name for data member and member function. The member function has same number of arguments, same data type.In this scenario, the derived class object may not able to access the content of base class. In order to overcome these drawbacks, a super keyword is used.

DEFINITION:

If Derived class has the same method or same data members as in base class,it is known as method overriding.This is nearly the same as method overloading,main difference is that overriding occurs at the runtime,overloading occurs at compile time.

DIFFERENCE BETWEEN METHOD OVERLOADING AND OVERRIDING:

METHOD OVERLOADING	METHOD OVERRIDING
It occurs within the same class.	It occurs between two classes. Ie.,base and derived class.
Inheritance is not involved.	Inheritance is involved.
One method does not hide another .	Derived class method hides the base class method.
Parameters must be different.	Parameters must be the same.
Return type may or maynot be the same.	Return type must be the same.

OCCURRENCE:

- **Occurs at data members-** Overriding occurs when the **data members** in Derived class is the same as in base class.
- **Occurs at member function-**Overriding occurs when the **member function** in Derived class is the same as in base class.

SYNTAX:

Class **Base**

```
{
     int data;                                  //Overridden method
    void display()
     {
          ……..
      }
}
Class Derived extends Base
{
    //Overriding method
    int data;  // same data member
    void display() // same member function
     {
        ……
      }
}
```

Note: In this derived class has the same data member and same member function as in base class. So,it is called method overriding.

EXAMPLE:

Method overriding at datamember:

```
class Base
{
    //int a=5;
    int a1=10;
    double a=10.5;//same data type
}
class Derived extends Base
{
    double a=20.5;// same as parent
   void display()
   {
      // output will be derived class a and base class a1
     // Derived class data member is overriding the base class
      System.out.println(a+" "+ a1);
    }
}
class Main
{
```

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

Output:

20.5 10

EXPLANATION:A base and derived class with the same data member **a** with data type called double are created. In main class ,create an object for derived class and call the display() method,the pointer goes to derived class and search for display() method in that current class.In that, print the **a** and **a1 values,** it prints Derived class **float a** value and base class **int a1** value as the output.Because, **a** is present in both the classes and a1 is in parent class. The priority is to print the current invoked class data.

Note: Here, Derived class takes the same data member **double a** as the base class. Note that,in base class there are two data member **a**(one of integer data type and another with the double data type). Overriding takes at the same data member with the same datatype.

Method overriding at member function:

```
class Base
{
  void display()// same member function
  {
     System.out.println("Parent class");
  }
}
class Derived extends Base
{
 void display()// same member function
 {
    System.out.println(" Child class");
 }
}
class Main
{
public static void main(String args[])
{
```

```
    Derived obj=new Derived();
     obj.display();
}
}
```

Output:
Child class.

EXPLANATION: A base and derived class with the same member function **display()** with same return type are created. In the main class , When an object for the derived class is created and calls the display() method,it goes to the derived class method and prints the **Child class**,since the object points to the derived class.

Note: Here member function in both Derived and base is the same .

In these two cases ,by using Derived class object we cannot be able to call the base class method.

To overcome this, there are two methods:

- Using **super** keyword.
- Using **dynamic method dispatch**.

SUPER KEYWORD:

- Super is the keyword, it is used for calling the parent class method or constructor.
- In this, we can use super keyword in two places:
    - Calling the data member and method- **super.** Used to call the data members and method of base class.
    - Calling constructor-**super()** Used to call the constructor of base class
- If in **multiple** and **multilevel** inheritance super keyword invokes the **immediate current** base class.

CONSTRAINS:

- Super can be used to refer to immediate parent class instance variables.
- Super can be used to invoke immediate parent class methods.
- Super with parameter parenthesis is used to invoke immediate parent class constructor.

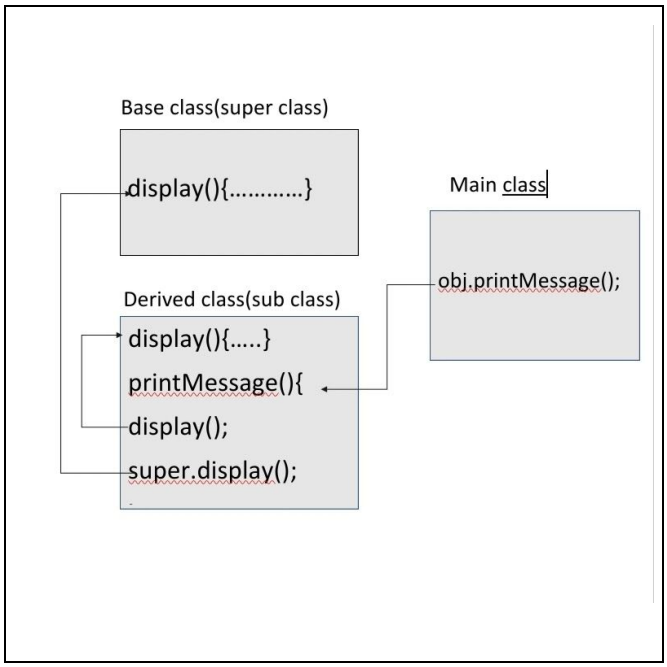
1.SUPER TO CALL BASE CLASS METHOD:

```
class Base
{
   //overridden method
   String a="parent class";
   void display()
   {
       System.out.println(a);
   }
}
class Derived extends Base
{
   //overriding method
   String a="child class";
    void display()
   {
      System .out.println(a);
    }
    void printMessage()
    {
      display();//calling the derived class method
      super.display();//calling the base class method
    }
}
class Main
{
  public static void main(String args[])
  {
    Derived obj=new Derived();
    obj.printMessage();
  }
}
```

Output:

child class
parent class

EXPLANATION:



2.ACCESS BASECLASS ATTRIBUTE:

```
class Base
{
   //overridden method
   String a="parent class";
  }
class Derived extends Base
{
   //overriding method
   String a="child class";
        void printMessage()
    {
       System .out.println(a);
       System .out.println(super.a);
    }
}
class Main
{
  public static void main(String args[])
  {
    Derived obj=new Derived();
    obj.printMessage();
  }
}
```

Output:
child class
parent class

EXPLANATION:
A parent and child class with the same data members **a** and **string** as its datatype is created . In main, create an **object** for the **derived** class and make a call to the printMessage() method. It invokes the derived class method. In that, print the value of the datamember **a** of derived class and also the parent class datamember **a** by using the **super.** keyword.

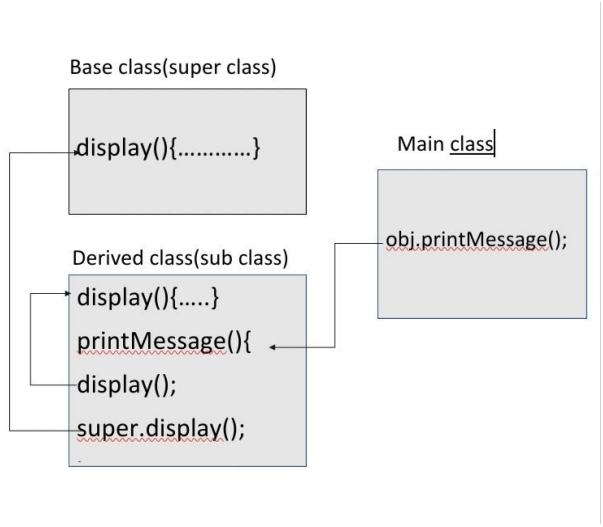
3.USE OF SUPER TO ACCESS BASE CLASS CONSTRUCTOR:

```
class Base
{
   //overridden method
   String a="parent class";
   Base()
   {
       System.out.println(a);
   }
}
class Derived extends Base
{
   //overriding method
   String a="child class";
    Derived()
   {
      super();
      System .out.println(a);
    }
}
class Main
{
  public static void main(String args[])
  {
    Derived obj=new Derived();
  }
}
```

Output:
parent class

child class

EXPLANATION:



4.CALL PARAMETERIZED CONSTRUCTOR USING SUPER:

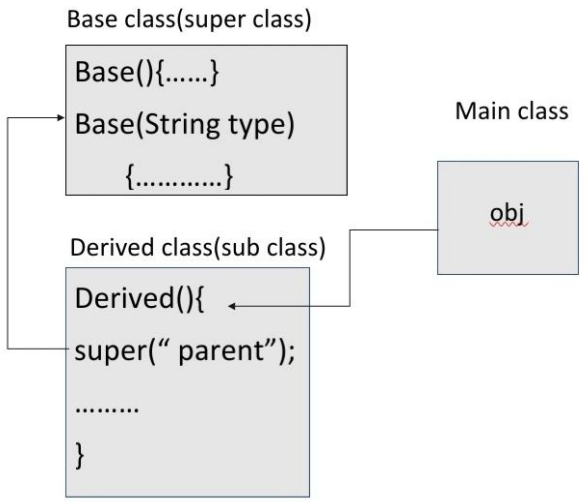
```
class Base
{
   //overridden method
   String a="parent class";
    Base()//constructor with no argument
   {
       System.out.println(a);
   }
   Base(String a)//constructor with arguments
   {
       System.out.println(a);
   }
}
class Derived extends Base
{
   //overriding method
   String a="child class";
    Derived()
   {
      super("parent class");//calling super class constructor with arguments
      System .out.println(a);
    }
}
```

```
class Main
{
  public static void main(String args[])
  {
    Derived obj=new Derived();
  }
}
```

Output:

parent class
child class

EXPLANATION:



DYNAMIC METHOD DISPATCH:

- Runtime polymorphism.
- The process of calling the base class method is resolved at runtime is known as dynamic method dispatch.
- In other words,when the base class reference variable points to the derived class during method call the derived class method is executed,this call is determined by the type of object at runtime.
- This method also is for calling the data members and member functions of both parent and child classes using reference variables.
-

EXAMPLE:

```
class Base
{
   //overridden method
   String a="parent class";
   void display()//same method
   {
       System.out.println(a);
   }
}
class Derived extends Base
{
   //overriding method
   String a="child class";
    void display()//same method
    {
       System .out.println(a);
    }
}
class Main
{
  public static void main(String args[])
  {
    Base obj1=new Base();   //creating object for base class
    obj1.display();   //calling method of base class using base object
   System.out.println("Base data:"+obj1.a);//calling parent data
    Derived dev=new Derived();  //creating object for derived class
    Base obj2;  //creating reference variable for base class
    obj2=dev;  //reference object points to the derived class
    obj2.display(); //calling derived class method using base reference object
     System.out.println("Derived data:"+obj2.a);//calling derived data
  }
}
```

Output:

parent class
child class

EXPLANATION:here base and derived class are created.in main, an object for both base class and for derived class are created, calling its method and accessing the base datamember using the base object within the main.In this dynamic method dispatch takes place at,when the reference object for base class is created and make it to points to the derived class,and make a call for the derived class method and accessing its data members.

How to use method overriding in Inheritance for subclasses ?

Solution:

This example demonstrates the way of method overriding by subclasses with different number and type of parameters.

Write a java program to find the area of the rectangle using method overriding. Also, identify the two approaches for accessing the base class contents.

```
class Figure // base class
{
  double dim1;
  double dim2;
  Figure(double a , double b) //parameterized constructor
  {
        dim1 = a;
        dim2 = b;
    }
  Double area()//same method
    {
       System.out.println("Inside area for figure.");
       return(dim1*dim2);
     }
}
class Rectangle extends Figure // derived class
{
   Rectangle(double a, double b) //parameterized constructor
  {
    super(a ,b);//using super keyword
  }
  Double area()//same method
  {
    System.out.println("Inside area for rectangle.");
    return(dim1*dim2);
  }
}
class Main
{
  public static void main (String []agrs)
  {
     Figure f = new Figure(10 , 10);//creating the object for constructor and passing arguments

     Rectangle r = new Rectangle(9 , 5);//creating the object for constructor and passing arguments
    Figure figref;  //reference variable of parent class
   figref = f;//make it to points to the base class
    System.out.println("Area is :"+figref.area());
```

```
    figref = r; ;//make it to points to the derived class
    System.out.println("Area is :"+figref.area());
   }
}
```

Output:

```
Inside area for figure.
Area is :100.0
Inside area for rectangle.
Area is :45.0
```

EXPLANATION:

Parent class: A parameter constructor and area() with double as its return type created in parent class name Figure.

Child class:A parameter constructor and area() same as parent class with double as its return type is created in child class named Rectangle.

Note that, in this example we used both **super and dynamic method dispatch** methods to implement **method overriding**.

In main,we created the **object** for the both parent and child classes,and **passed an arguments**.while creating an object for child class,since it is a constructor,passed arguments should be **assigned** to the **base** class while creating the object itself. Call to the parameterized constructor using **super() keyword**.

Thus we assign the values to the **data members** of the **base** class using **super keyword**.

Then in main we have created a parent class **reference variable** . Then we made it to **point** to the **base** class and **derived** class,thus **base** class and **derived** class **method** can be accessed through it respectively.That is,**parent** class object **f** is assigned to reference object **figref**,and we make a call **figref.area()**, it returns the value from the **parent** class.While,**child class** object **r** is assigned to reference object **figref**,and we make a call **figref.area()**, it returns the value from the **child class**.