Super keyword in java with example

BY CHAITANYA SINGH | FILED UNDER: OOPS CONCEPT

The super keyword refers to the objects of immediate parent class. Before learning super keyword you must have the knowledge of inheritance in Java so that you can understand the examples given in this guide.

The use of super keyword

1) To access the data members of parent class when both parent and child class have member with same name
2) To explicitly call the no-arg and parameterized constructor of parent class
3) To access the method of parent class when child class has overridden that method.

Now lets discuss them in detail with the help of examples:

1) How to use super keyword to access the variables of parent class

When you have a variable in child class which is already present in the parent class then in order to access the variable of parent class, you need to use the super keyword.

Lets take an example to understand this: In the following program, we have a data member `num`declared in the child class, the member with the same name is already present in the parent class. There is no way you can access the `num` variable of parent class without using super keyword. .

```
//Parent class or Superclass or base class
class Superclass
{
   int num = 100;
}
//Child class or subclass or derived class
class Subclass extends Superclass
{
   /* The same variable num is declared in the Subclass
    * which is already present in the Superclass
    */
    int num = 110;
    void printNumber(){
        System.out.println(num);
    }
    public static void main(String args[]){
        Subclass obj= new Subclass();
```

```
        obj.printNumber();
    }
}
```
Output:
110

Accessing the num variable of parent class:
By calling a variable like this, we can access the variable of parent class if both the classes (parent and child) have same variable.

```
super.variable_name
```
Let's take the same example that we have seen above, this time in print statement we are passing `super.num` instead of num.

```
class Superclass
{
   int num = 100;
}
class Subclass extends Superclass
{
   int num = 110;
   void printNumber(){
        /* Note that instead of writing num we are
         * writing super.num in the print statement
         * this refers to the num variable of Superclass
         */
        System.out.println(super.num);
   }
   public static void main(String args[]){
        Subclass obj= new Subclass();
        obj.printNumber();
   }
}
```
Output:
100
As you can see by using super.num we accessed the num variable of parent class.

2) Use of super keyword to invoke constructor of parent class

When we create the object of sub class, the new keyword invokes the constructor of child class, which implicitly invokes the constructor of parent class. So the order to execution when we create the object of child class is: parent class constructor is executed first and then the child class constructor is executed. It happens because compiler itself adds super()(this invokes the no-arg constructor of parent class) as the first statement in the constructor of child class.

Let's see an example to understand what I have explained above:

```
class Parentclass
{
   Parentclass(){
        System.out.println("Constructor of parent class");
   }
}
class Subclass extends Parentclass
{
   Subclass(){
        /* Compile implicitly adds super() here as the
         *  first statement of this constructor.
         */
        System.out.println("Constructor of child class");
   }
   Subclass(int num){
        /* Even though it is a parameterized constructor.
         * The compiler still adds the no-arg super() here
         */
        System.out.println("arg constructor of child class");
   }
   void display(){
        System.out.println("Hello!");
   }
   public static void main(String args[]){
        /* Creating object using default constructor. This
         * will invoke child class constructor, which will
         * invoke parent class constructor
         */
        Subclass obj= new Subclass();
        //Calling sub class method
        obj.display();
        /* Creating second object using arg constructor
         * it will invoke arg constructor of child class which will
         * invoke no-arg constructor of parent class automatically
         */
        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 to invoke parameterized constructor of parent class

We can call super() explicitly in the constructor of child class, but it would not make any sense because it would be redundant. It's like explicitly doing something which would be implicitly done otherwise.

However when we have a constructor in parent class that takes arguments then we can use parameterized super, like `super(100);` to invoke parameterized constructor of parent class from the constructor of child class.
Let's see an example to understand this:

```
class Parentclass
{
   //no-arg constructor
   Parentclass(){
        System.out.println("no-arg constructor of parent class");
   }
   //arg or parameterized constructor
   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. Another important
        * point to note is that when we explicitly use super in constructor
        * the compiler doesn't invoke the parent constructor automatically.
        */
        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
```

There are few important points to note in this example:
1) super()(or parameterized super must be the first statement in constructor otherwise you will get the compilation error: "Constructor call must be the first statement in a constructor"
2) When we explicitly placed super in the constructor, the java compiler didn't call the default no-arg constructor of parent class.

3) How to use super keyword in case of method overriding

When a child class declares a same method which is already present in the parent class then this is called method overriding. We will learn method overriding in the next tutorials of this series. For now you just need to remember this: When a child class overrides a method of parent class, then the call to the method from child class object always call the child class version of the method. However by using super keyword like this: super.method_name you can call the method of parent class (the method which is overridden). In case of method overriding, these terminologies are used: Overridden method: The method of parent class Overriding method: The method of child class Lets take an example to understand this concept:

```
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(){
        //This would call Overriding method
        display();
        //This would call Overridden method
        super.display();
   }
   public static void main(String args[]){
        Subclass obj= new Subclass();
        obj.printMsg();
   }
}
```

Output:

```
Child class method
Parent class method
```

What if the child class is not overriding any method: No need of super

When child class doesn't override the parent class method then we don't need to use the super keyword to call the parent class method. This is because in this case we have only one version of each method and child class has access to the parent class methods so we can directly call the methods of parent class without using super.

```
class Parentclass
{
   void display(){
```

```
        System.out.println("Parent class method");
   }
}
class Subclass extends Parentclass
{
   void printMsg(){
        /* This would call method of parent class,
         * no need to use super keyword because no other
         * method with the same name is present in this class
         */
        display();
   }
   public static void main(String args[]){

        Subclass obj= new Subclass();
        obj.printMsg();
   }
}
```

Output:

```
Parent class method
```