Super keyword in Java

The `super` keyword in Java is used to interact with a parent class. It serves several purposes, such as calling parent class constructors, accessing parent class methods, and resolving conflicts between members in a parent and child class.

Uses of `super` Keyword:

1. Access Parent Class Constructor:

- You can use `super()` to call the constructor of a parent class from a child class constructor.
- The `super` keyword is useful when a child class needs to initialize values using the constructor of the parent class.

2. Access Parent Class Data Members and Methods: In cases where a child class and parent class have members with the same name, `super` can help access the parent class's members directly.

3. Resolve Name Conflicts: When a variable or method in a child class has the same name as one in the parent class, `super` can resolve the conflict by specifically referencing the parent class's member.

4. Method Overriding: In method overriding, where both the parent and child classes have methods with the same name, `super` can be used to call the parent class's method inside the child class.

Example: Using `super` to Call Parent Class Constructors

Consider the following code that demonstrates how the `super` keyword works to call the parent class constructor:

```java
class Base {
    int x;
    Base() {
        x = 0;
    }

    Base(int i) {
        x = i;
    }
}

class Derived extends Base {
    int y;

    Derived() {
        super();  // Calls Base class constructor with no parameters
        y = 0;
    }

    Derived(int i, int j) {
        super(i);  // Calls Base class constructor with one parameter
        y = j;
    }
}

class Test {
    public static void main(String[] args) {
        Derived d = new Derived(10, 20);
        System.out.println(d.x);
        System.out.println(d.y);
    }
}
```

Output:

```
10
20
```

In this example:

- The `Base` class has two constructors: a parameterless constructor that sets `x` to 0, and a parameterized constructor that sets `x` to a given value.
- The `Derived` class calls the `super()` constructor of the `Base` class to initialize `x`.
- The `Test` class creates an instance of `Derived`, which initializes `x` to 10 and `y` to 20.

Constructor Behavior in `super`

- Even if you do not explicitly use `super()` in the child class constructor, the default constructor of the parent class is automatically called. This means if the parent class has a parameterless constructor, it will still be called, and the behavior of the program will remain consistent.
- If you want to call a parameterized constructor in the parent class, you can explicitly use `super()` with arguments.

Example: Avoiding Ambiguity with Same Variable Names

If both the parent and child class have a variable with the same name, using `super` helps to resolve the ambiguity. Here's an example:

```java
class Base {
    int x = 10;
}

class Derived extends Base {
    int x = 20;

    void print() {
        System.out.println(super.x);  // Accesses parent class's x
        System.out.println(x);         // Accesses child class's x
    }
}

class Test {
    public static void main(String[] args) {
        Derived d = new Derived();
        d.print();
    }
}

```

Output:

```
10
20
```

In this case:

- The `Base` class has a variable `x` initialized to 10.
- The `Derived` class also has a variable `x` initialized to 20.
- The `print()` method uses `super.x` to access the `x` variable of the parent class, and `x` alone refers to the `x`variable in the child class.