

24-this and super method

this and super Methods in Java

In Java, the super() and this() keywords are crucial for managing constructor invocation and establishing relationships between classes. Understanding how to use these keywords effectively is vital for mastering object-oriented programming in Java.

this Keyword

The this keyword is a **reserved keyword** in Java that refers to the current instance of a class. It is primarily used for calling one constructor from another within the same class. Here are some contexts where this can be used:

- To invoke the current class constructor: This is commonly used when you want to call one constructor from another within the same class.
- To pass the current class instance as an argument: this can be passed as an argument in method or constructor calls.

Example:

Consider a scenario where class B has both a default constructor and a parameterized constructor. If you want both constructors to be executed when the parameterized constructor is called, you can use the this keyword to call the default constructor from the parameterized one.

```
class A {
 1
         public A() {
             System.out.println("In A's Constructor");
 3
 4
 5
    }
 6
    class B extends A {
 8 -
         public B() {
 9
             super(); // This is present by default
10
             System.out.println("In B's Constructor");
11
12
13
         public B(int n) {
14
             this(); // Calls the default constructor of the current class
15
             System.out.println("In B's parameterized constructor: " + n);
16
         }
17
18
19 public class Demo {
20 -
         public static void main(String[] args) {
21
             B \text{ obj} = \text{new B(10)};
22
23
```





Output:

```
Output Generated Files

In A's Constructor
In B's Constructor
In B's parameterized constructor: 10
```

super Keyword

The super keyword is another reserved keyword in Java, used to refer to the superclass (parent class) of the current object. It has several important uses:

- To call superclass constructors: super() is often used to call a superclass constructor from a subclass. If a superclass has a parameterized constructor, you can call it explicitly using super().
- To refer to superclass members: super can also be used to refer to fields, methods, or constructors of the superclass, which is particularly useful when the subclass has members with the same names as those in the superclass.

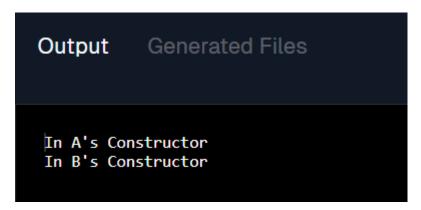
Example:

```
class A {
         public A() {
 3
             System.out.println("In A's Constructor");
 4
 5
 6
 7 -
    class B extends A {
 8 -
        public B() {
             super(); // Calls the superclass (A's) constructor
9
             System.out.println("In B's Constructor");
10
11
         }
12
    }
13
    public class Demo {
14
         public static void main(String[] args) {
15 -
             B obj = new B();
16
17
18
```





Output:



In the above example, when an object of class B is created, the constructor of A is called first, followed by the constructor of B. This happens because the super() keyword, which calls the parent class constructor, is implicitly included in every constructor.

Important Points to Remember

- Implicit super(): Every constructor in Java implicitly contains a call to super() unless you explicitly call another constructor using this(). This ensures that the parent class constructor is always invoked before the subclass constructor.
- **this() vs. super()**: Both this() and super() must be the first statement in a constructor. You cannot use both in the same constructor, as they would conflict in terms of order.
- The Object Class:
 - o In Java, all classes implicitly inherit from the Object class if they don't explicitly extend another class. This makes Object the root of the class hierarchy.
 - o When a class like A is extended by class B, A itself extends the Object class. This means that B also indirectly inherits from Object. Therefore, every class in Java, either directly or indirectly, inherits from the Object class.
- **No Multiple Inheritance**: Java does not support multiple inheritance for classes, which avoids complexity and ambiguity, especially with the super keyword. However, Java supports multilevel inheritance, where super is still relevant.

Example of Error with this() and super() Together:





```
class A {
 2 -
        A() {
 3
             System.out.println("A is created.");
 4
         }
 5
    }
 7 -
    class B extends A {
8 -
        B() {
9
             System.out.println("B is created.");
10
11
12 -
        B(String name) {
13
             super(); // Calls the parent class constructor
             this(); // Error: Cannot call this() after super()
14
15
             System.out.println("B name is " + name);
        }
16
17
    }
18
19 public class Demo {
20 -
        public static void main(String args[]) {
21
             B obj = new B("Telusko");
22
23
24
```

Output:

Compile Error: "call to this must be first statement in constructor"

```
Output Generated Files

Demo.java:14: error: call to this must be first statement in constructor this(); // Error: Cannot call this() after super()

1 error
```

Commonly Asked Questions

- Can we use both this() and super() in the same constructor?
 - o No, in Java, either this() or super() must be the first statement in a constructor. You cannot use both together, as it would create a conflict, leading to a compile-time error.
- What happens if super() is not explicitly mentioned in a constructor?





o If super() is not explicitly mentioned, Java implicitly calls the default constructor of the superclass.

