

[Courses](#)[Login](#)[Write an Article](#)

Difference between super() and this() in java

Similar article : [super and this keyword](#)

super() as well as this() both are used to make **constructor calls**. super() is used to call **Base** class's constructor(i.e, Parent's class) while this() is used to call **current** class's constructor.

Let's see both of them in detail:

super()

1. **super()** is use to call Base class's(Parent class's) constructor.

// Java code to illustrate usage of super()

```
class Parent {
    Parent()
    {
        System.out.println("Parent class's No " +
                           " arg constructor");
    }
}

class Child extends Parent {
    Child()
    {
        super();
        System.out.println("Flow comes back from " +
                           "Parent class no arg const");
    }
    public static void main(String[] args)
    {
        new Child();
        System.out.println("Inside Main");
    }
}
```

Output:



Parent class's No arg constructor

Flow comes back from Parent class no arg const

Inside Main

Flow of Program :



- In main, we have made a statement **new Child()**, so it calls the no argument constructor of Child class.
 - Inside that we have **super()** which calls the no argument of Parent class since we have written **super()** and no arguments that's why it calls no argument constructor of Parent class, in that we have an SOP statement and hence it prints *Parent class's No arg constructor*.
 - Now as the No argument const of Parent class completes so flow comes back to the no argument of the Child class and in that we have an SOP statement and hence it prints *Flow comes back from Parent class no arg const*.
 - Further after completing the no argument constructor of child class flow now came back again to main and executes remaining statements and prints *Inside Main*.
2. We can use **super()** **only inside constructor and nowhere else**, not even in static context not even inside methods and **super()** should be **first statement** inside constructor.

```
// Java program to illustrate usage of
// super() as first statement

class Parent {
    Parent()
    {
        System.out.println("Parent class's No " +
                           "arg constructor");
    }
}

class Child extends Parent {
    Child()
    {
        // Uncommenting below line causes compilation
        // error because super() should be first statement
        // System.out.println("Compile Time Error");
        super();

        System.out.println("Flow comes back from " +
```



```

        "Parent class no arg const");
    }

    public static void main(String[] args)
    {
        new Child();
        System.out.println("Inside main");
    }
}

```

Output:

```

Parent class's No arg constructor
Flow comes back from Parent class no arg const
Inside main

```

Note : super() should be **first** statement inside any constructor. It can be used **only inside constructor** and nowhere else. super() is used to refer **only parent class's(super class's) constructor**.

this()

1. this() is used to call **current class's constructor**.

```

// Java code to illustrate usage of this()

class RR {
    RR()
    {
        this(10);
        System.out.println("Flow comes back from " +
                           "RR class's 1 arg const");
    }

    RR(int a)
    {
        System.out.println("RR class's 1 arg const");
    }
    public static void main(String[] args)
    {
        new RR();
        System.out.println("Inside Main");
    }
}

```

Output:

```

RR class's 1 arg const
Flow comes back from RR class's 1 arg const
Inside Main

```



Flow of Program :

- First start from main and then in that we have made a statement **new Child()** hence which calls the no argument constructor of Child class, inside that we have **this(10)** which calls the 1 argument of current class(i.e, RR class)
 - Since we have written this(10) and 1 argument that's why it calls 1 argument constructor of RR class. In that we have an SOP statement and hence it prints *RR class's 1 arg const.*
 - Now as the 1 argument const of RR class completes so flow comes back to the no argument of the RR class and in that we have an SOP statement and hence it prints *Flow comes back from RR class's 1 arg const.*
 - Further after completing the no argument constructor of RR class flow now came back again to main and executes remaining statements and prints *Inside Main.*
2. We can use this() **only inside constructor and nowhere else**, not even in static context not even inside methods and this() should be **first statement** inside constructor.

```
// Java program to illustrate usage of
// this() as first statement

class RR {
    RR()
    {
        // Uncommenting below line causes compilation
        // error because this() should be first statement
        // System.out.println("Compile Time Error");
        this(51);
        System.out.println("Flow comes back from RR " +
                           "class 1 arg const");
    }
    RR(int k)
    {
        System.out.println("RR class's 1 arg const");
    }
    public static void main(String[] args)
    {
        new RR();
        System.out.println("Inside main");
    }
}
```

Output:

```
RR class's 1 arg constructor
Flow comes back from RR class 1 arg const
Inside main
```



Note : this() should be **first** statement inside any constructor. It can be used **only inside constructor** and nowhere else. this() is use to refer **only current class's constructor**.

Important points about this() and super()

1. We can use super() as well this() **only once** inside constructor. If we use super() twice or this() twice or super() followed by this() or this() followed by super(), then immediately we get compile time error i.e, we can use **either super() or this() as first statement inside constructor and not both**.
2. It is upto you that whether you use super() or this() or not because if we are not using this() or super() then **by default compiler will put super()** as first statement inside constructor.

```
// Java program to illustrate super() by default
// executed by compiler if not provided explicitly

class Parent {
    Parent()
    {
        System.out.println("Parent class's No " +
                           "argument constructor");
    }
    Parent(int a)
    {
        System.out.println("Parent class's 1 argument" +
                           " constructor");
    }
}

class Base extends Parent {
    Base()
    {
        // By default compiler put super()
        // here and not super(int)
        System.out.println("Base class's No " +
                           "argument constructor");
    }
    public static void main(String[] args)
    {
        new Base();
        System.out.println("Inside Main");
    }
}
```

Output:

Parent class's No argument constructor
Base class's No argument constructor
Inside Main



Flow of program:

- Inside main we have **new Base()** then flow goes to **No argument constructor** of Base class.
- After that if we don't put either super() or this() then **by default compiler put super()**.
- So flow goes to **Parent class's No arg constructor and not 1 argument constructor**.
- After that it prints *Parent class's No argument constructor*.
- After that when Parent() constructor completes then flow again **comes back** to that **No argument constructor of Base class** and executes next SOP statement i.e, *Base class's No argument constructor*.
- After completing that No argument constructor flow comes **back to main()** again and prints the remaining statements inside main() i.e, *Inside main*

However, if explicitly specified, you may use this() before super.

```
// Java program to illustrate super() put by
// compiler always if not provided explicitly
```

```
class Parent {
    Parent()
    {
        System.out.println("Parent class's No " +
                           "argument constructor");
    }
    Parent(int a)
    {
        System.out.println("Parent class's one " +
                           "argument constructor");
    }
}

class Base extends Parent {
    Base()
    {
        this(10);
        System.out.println("No arg const");
    }
    Base(int a)
    {
        this(10, 20);
        System.out.println("1 arg const");
    }
    Base(int k, int m)
    {
        // See here by default compiler put super();
        System.out.println("2 arg const");
    }
    public static void main(String[] args)
    {
        new Base();
        System.out.println("Inside Main");
    }
}
```



```
    }
}
```

Output:

```
Parent class's No argument constructor
2 arg const
1 arg const
No arg const
Inside Main
```

3. Recursive constructor call not allowed

```
// Java program to illustrate recursive
// constructor call not allowed

class RR {
    RR()
    {
        this(30);
    }
    RR(int a)
    {
        this();
    }
    public static void main(String[] args)
    {
        new RR();
    }
}
```



Output:

```
Compile time error saying recursive constructor invocation
```

Flow of program : Here, above start from main() and then flow goes to **No arg constructor of RR class**. After that we have **this(30)** and flow goes to **1 arg constructor of RR** and in that we have **this()** so again flow goes to No arg constructor of base class and in that again we have this(30) and flow again goes to 1 arg constructor of Base class and **it goes on** like a **recursion**. So it is invalid that's why we **get compile time error** saying *recursive constructor invocation*. So recursive constructor invocations are **not allowed in java**.

This article is contributed by **Rajat Rawat**. If you like GeeksforGeeks and would like to contribute, you can also write an article using [contribute.geeksforgeeks.org](https://www.geeksforgeeks.org/contribute) or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

 **Top IT MNCs hiring freshers - CTC:25k-5Lac/**
 Engineering, BCA, MCA, BSc Graduates & Diploma Holders c

Recommended Posts:

Super Keyword in Java

super and this keywords in Java

Accessing Grandparent's member in Java using super

Difference between x++ and x=x+1 in Java

Difference between Java and C language

Difference between Java and JavaScript

Difference between notify() and notifyAll() in Java

Difference between ArrayList and HashSet in Java

Difference between concat() and + operator in Java

Difference between throw and throws in Java

Difference between == and .equals() method in Java

Difference between print() and println() in Java

Difference between Arrays and Collection in Java

Difference between an Integer and int in Java with Examples

Difference between an Iterator and ListIterator in Java

1 **Top IT MNCs hiring freshers - CTC:25k-5Lac/Month Exp:3-15+**

2 **Download PDF (Free)** [To View PDF, Download Here FromDocToPDF](#)

Article Tags : [Difference Between](#) [Java](#)

Practice Tags : [Java](#)



1

☐ To-do ☐ Done

4.2



Based on 4 vote(s)