Difference between “this” and“super” keywords in Java

Ask Question

What is the difference between the keywords `this` and `super` ?

Both are used to access constructors of class right? Can any of you explain?

java keyword

edited Jul 31 '13 at 20:23
informatik01
12.4k 8 54 85

asked Oct 26 '10 at 11:52
Sumithra
2,414 14 41 46

11 Answers

Lets consider this situation

```
class Animal {
  void eat() {
    System.out.println("animal : eat");
  }
}

class Dog extends Animal {
  void eat() {
    System.out.println("dog : eat");
  }
  void anotherEat() {
    super.eat();
  }
}

public class Test {
  public static void main(String[] args) {
    Animal a = new Animal();
    a.eat();
    Dog d = new Dog();
    d.eat();
    d.anotherEat();
  }
}
```

The output is going to be

```
animal : eat
dog : eat
animal : eat
```

The third line is printing "animal:eat" because we are calling `super.eat()`. If we called `this.eat()`, it would have printed as "dog:eat".

answered Oct 26 '10 at 12:00
Nithesh Chandra
1,672 12 12

2 I don't find this answer confusing, though you can make the last line of the output bold or add a trailing comment to emphasize that the base class was used. – razzak Apr 1 '16 at 7:13

`super` is used to access methods of the base class while `this` is used to access methods of the current class.

Extending the notion, if you write `super()`, it refers to constructor of the base class, and if you write `this()`, it refers to the constructor of the very class where you are writing this code.

edited Oct 26 '10 at 12:20

answered Oct 26 '10 at 11:53

Jaywalker
2,044 2 21 37

`this` is a reference to the object typed as the current class, and `super` is a reference to the object typed as its parent class.

In the constructor, `this()` calls a constructor defined in the current class. `super()` calls a constructor defined in the parent class. The constructor may be defined in any parent class, but it will refer to the one overridden closest to the current class. Calls to other constructors in this way may only be done as the first line in a constructor.

Calling methods works the same way. Calling `this.method()` calls a method defined in the current class where `super.method()` will call the same method as defined in the parent class.

answered Oct 26 '10 at 11:57

Erick Robertson
23.4k 7 60 92

Nice explanation! Clear and Concise! – Keith May 1 '17 at 11:50

From your question, I take it that you are really asking about the use of `this` and `super` in constructor chaining; e.g.

```
public class A extends B {
    public A(...) {
        this(...);
        ...
    }
}
```

versus

```
public class A extends B {
    public A(...) {
        super(...);
        ...
    }
}
```

The difference is simple:

- The `this` form chains to a constructor in the current class; i.e. in the `A` class.
- The `super` form chains to a constructor in the immediate superclass; i.e. in the `B` class.

answered Oct 26 '10 at 12:03
Stephen C
486k 66 536 882

`this` refers to a reference of the **current** class.
`super` refers to the **parent** of the current class (which called the `super` keyword).

By doing `this`, it allows you to access methods/attributes of the current class (including its own private methods/attributes).

`super` allows you to access public/protected method/attributes of parent(base) class. You cannot see the parent's private method/attributes.

edited Sep 8 '14 at 12:52

answered Oct 26 '10 at 11:54
Buhake Sindi
67.7k 22 142 201

2 This is answer is right if you change every occurrence of 'class' into 'object'. It is for instance not possible to call 'this' from a static method within a class. – Dave Oct 26 '10 at 13:09

@Dave, true...I basically went on the fact that super calls the base class (since it's a derived class of a base class). Should I say base object? If so, what's the difference between class/object? – Buhake Sindi Oct 26 '10 at 13:22

@TEG, I know it is a bit juggling with words and a lot of people use class and object as synonyms. The class is in fact the definition and may have static methods, constants and may even not have the possibility to be instantiated (abstract classes). An object can only exist at runtime and must be created with the ´new´ keyword. – Dave Oct 26 '10 at 13:53

@Dave, true, but if you look at literature, you will see words such as `base` and `derived` classes and not `based` and `derived` objects. Maybe the new literature distinguished the difference. – Buhake Sindi Oct 26 '10 at 14:05

@TEG, I agree on the usage of 'base' and 'derived' classes in context of a class diagram (or technical analysis) as more informal naming for superclass and subclass respectively. – Dave Oct 27 '10 at 7:43

`this` is used to access the methods and fields of the current object. For this reason, it has no meaning in static methods, for example.

`super` allows access to non-private methods and fields in the super-class, and to access constructors from within the class' constructors only.

answered Oct 26 '10 at 11:55

David Rabinowitz
21.9k 12 75 110

When writing code you generally don't want to repeat yourself. If you have an class that can be constructed with various numbers of parameters a common solution to avoid repeating yourself is to simply call another constructor with defaults in the missing arguments. There is only one annoying restriction to this - it must be the first line of the declared constructor. Example:

```
MyClass()
{
   this(default1, default2);
}

MyClass(arg1, arg2)
{
   validate arguments, etc...
   note that your validation logic is only written once now
}
```

As for the `super()` constructor, again unlike `super.method()` access it must be the first line of your constructor. After that it is very much like the `this()` constructors, DRY (Don't Repeat Yourself), if the class you extend has a constructor that does some of what you want then use it and then continue with constructing your object, example:

```
YourClass extends MyClass
{
   YourClass(arg1, arg2, arg3)
   {
      super(arg1, arg2) // calls MyClass(arg1, arg2)
      validate and process arg3...
   }
}
```

Additional information:

Even though you don't see it, the default no argument constructor always calls `super()` first. Example:

```
MyClass()
{
}
```

is equivalent to

```
MyClass()
{
   super();
}
```

I see that many have mentioned using the `this` and `super` keywords on methods and variables - all good. Just remember that constructors have unique restrictions on their usage, most notable is that they must be the very first instruction of the declared constructor and you can only use one.

answered Oct 26 '10 at 12:28

BigMac66

833 4 12 30

---

this keyword use to call constructor in the same class (other overloaded constructor)

syntax: **this** (args list); //compatible with **args list** in other constructor in the same class

super keyword use to call constructor in the super class.

syntax: super (args list); //compatible with **args list** in the constructor of the super class.

Ex:

```
public class Rect {
int x1, y1, x2, y2;

public Rect(int x1, int y1, int x2, int y2) // 1st constructor
{ ....//code to build a rectangle }
}
```

```
public Rect () {   // 2nd constructor
this (0,0,width,height) // call 1st constructor (because it has **4 int args**),
this is another way to build a rectangle
}


public class DrawableRect extends Rect {

public DrawableRect (int a1, int b1, int a2, int b2) {
super (a1,b1,a2,b2) // call super class constructor (Rect class)
}
}
```

edited Aug 11 '13 at 5:17

answered Aug 11 '13 at 4:32
rocketmanu
57 1 2 10

---

super() & this()

- super() - to call parent class constructor.
- this() - to call same class constructor.

NOTE:

- We can use super() and this() only in constructor not anywhere else, any attempt to do so will lead to compile-time error.
- We have to keep either super() or this() as the first line of the constructor but NOT both simultaneously.

super & this keyword

- super - to call parent class members(variables and methods).
- this - to call same class members(variables and methods).

NOTE: We can use both of them anywhere in a class except static areas(static block or method), any attempt to do so will lead to compile-time error.

edited Nov 3 '17 at 14:50

answered Nov 3 '17 at 14:20
Varun Vashista

69 5

1. use of SUPER keyword.

If your method overrides one of its superclass's methods, you can invoke the overridden method through the use of the keyword super. You can also use super to refer to a hidden field (although hiding fields is discouraged). Consider this class, Superclass:

```java
public class Superclass {

public void printMethod() {
    System.out.println("Printed in Superclass.");
}

}
```

Here is a subclass, called Subclass, that overrides printMethod():

```java
public class Subclass extends Superclass {

// overrides printMethod in Superclass
public void printMethod() {
    super.printMethod();
    System.out.println("Printed in Subclass");
}
public static void main(String[] args) {
    Subclass s = new Subclass();
    s.printMethod();
}

}
```

Within Subclass, the simple name printMethod() refers to the one declared in Subclass, which overrides the one in Superclass. So, to refer to printMethod() inherited from Superclass, Subclass must use a qualified name, using super as shown. Compiling and executing Subclass prints the following:

Printed in Superclass. Printed in Subclass.

2. use of THIS keyword **Using this with a Field**

The most common reason for using the this keyword is because a field is shadowed by a method or constructor parameter.

For example, the Point class was written like this

```
public class Point {
public int x = 0;
public int y = 0;

//constructor
public Point(int a, int b) {
    x = a;
    y = b;
}

}
```

but it could have been written like this:

```
public class Point {
public int x = 0;
public int y = 0;

//constructor
public Point(int x, int y) {
    this.x = x;
    this.y = y;
}

}
```

Each argument to the constructor shadows one of the object's fields — inside the constructor x is a local copy of the constructor's first argument. To refer to the Point field x, the constructor must use this.x.

Using this with a Constructor

From within a constructor, you can also use the this keyword to call another constructor in the same class. Doing so is called an explicit constructor invocation. Here's another Rectangle class, with a different implementation from the one in the Objects section.

```
public class Rectangle {
private int x, y;
private int width, height;

public Rectangle() {
        this(0, 0, 0, 0);
    }
    public Rectangle(int width, int height) {
        this(0, 0, width, height);
    }
    public Rectangle(int x, int y, int width, int height) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
    }

}
```

This class contains a set of constructors. Each constructor initializes some or all of the rectangle's member variables. The constructors provide a default value for any member variable whose initial value is not provided by an argument. For example, the no-argument constructor calls the four-argument constructor with four 0 values and the two-argument constructor calls the four-argument constructor with two 0 values. As before, the compiler determines which constructor to call, based on the number and the type of arguments.

If present, the invocation of another constructor must be the first line in the constructor.

answered Nov 8 '13 at 7:54
nagSumanth
71 9

This almost appears to be a situation where a person has asked a question that everyone sees only one possible answer for. Yet the person calls each answer a non-answer, so everyone tries a different variant on the same answer since there appears to be only one way to answer it.

So, one person starts out with A is a class and B is its subclass. The next person starts out with C is a Class and B is its subclass. A third person answers it a little differently. A is a class and B extends A. Then a fourth one says A is A class that gets extended as B is defined. Or Animal is a class and Dog is a subclass. Or Nation is a class and China is a special case of Nation.

Or better yet, Man is a class, and Clark Kent is a subclass of Man. So, Superman...no...that doesn't work in terms of Java....

Well, it's at least another attempt to come up with a different description that nobody came up with that might explain things differently. Arggghhhh.

It seems to me that everyone's explanation worked except mine.

answered Sep 9 '15 at 22:38

Dan
9

Please use the appropriate area for comments, if your intentions are not to answer the question. – zer00ne Sep 9 '15 at 22:53

@Dan you will be able to add comments once your reputation score is a little higher, you are likely to get downvoted for including comments in an answer- you can of course edit your answer at any time – Mousey Sep 10 '15 at 1:08