**Java super**

The super keyword in Java is used in subclasses to access superclass members (attributes, constructors and methods).

Before we learn about the super keyword, make sure to know about [Java inheritance](https://www.programiz.com/java-programming/inheritance).

**Uses of super keyword**

1. To call methods of the superclass that is overridden in the subclass.
2. To access attributes (fields) of the superclass if both superclass and subclass have attributes with the same name.
3. To explicitly call superclass no-arg (default) or parameterized constructor from the subclass constructor.

Let’s understand each of these uses.

**1. Access Overridden Methods of the superclass**

If methods with the same name are defined in both superclass and subclass, the method in the subclass overrides the method in the superclass. This is called [method overriding](https://www.programiz.com/java-programming/method-overriding).

**Example 1: Method overriding**

class Animal {

// overridden method

public void display(){

System.out.println("I am an animal");

}

}

class Dog extends Animal {

// overriding method

@Override

public void display(){

System.out.println("I am a dog");

}

public void printMessage(){

display();

}

}

class Main {

public static void main(String[] args) {

Dog dog1 = new Dog();

dog1.printMessage();

}

}

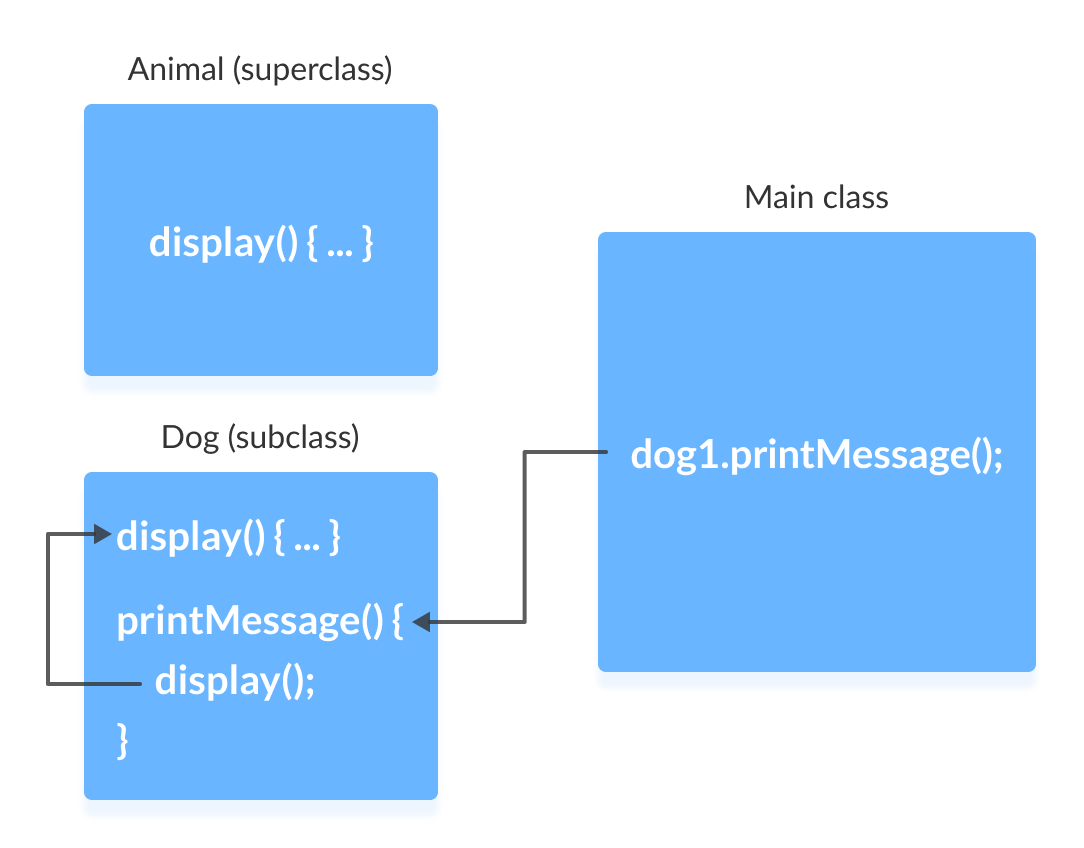
[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**

I am a dog

In this example, by making an object dog1 of Dog class, we can call its method printMessage() which then executes the display() statement.

Since display() is defined in both the classes, the method of subclass Dog overrides the method of superclass Animal. Hence, the display() of the subclass is called.



**What if the overridden method of the superclass has to be called?**

We use super.display() if the overridden method display() of superclass Animal needs to be called.

**Example 2: super to Call Superclass Method**

class Animal {

// overridden method

public void display(){

System.out.println("I am an animal");

}

}

class Dog extends Animal {

// overriding method

@Override

public void display(){

System.out.println("I am a dog");

}

public void printMessage(){

// this calls overriding method

display();

// this calls overridden method

super.display();

}

}

class Main {

public static void main(String[] args) {

Dog dog1 = new Dog();

dog1.printMessage();

}

}

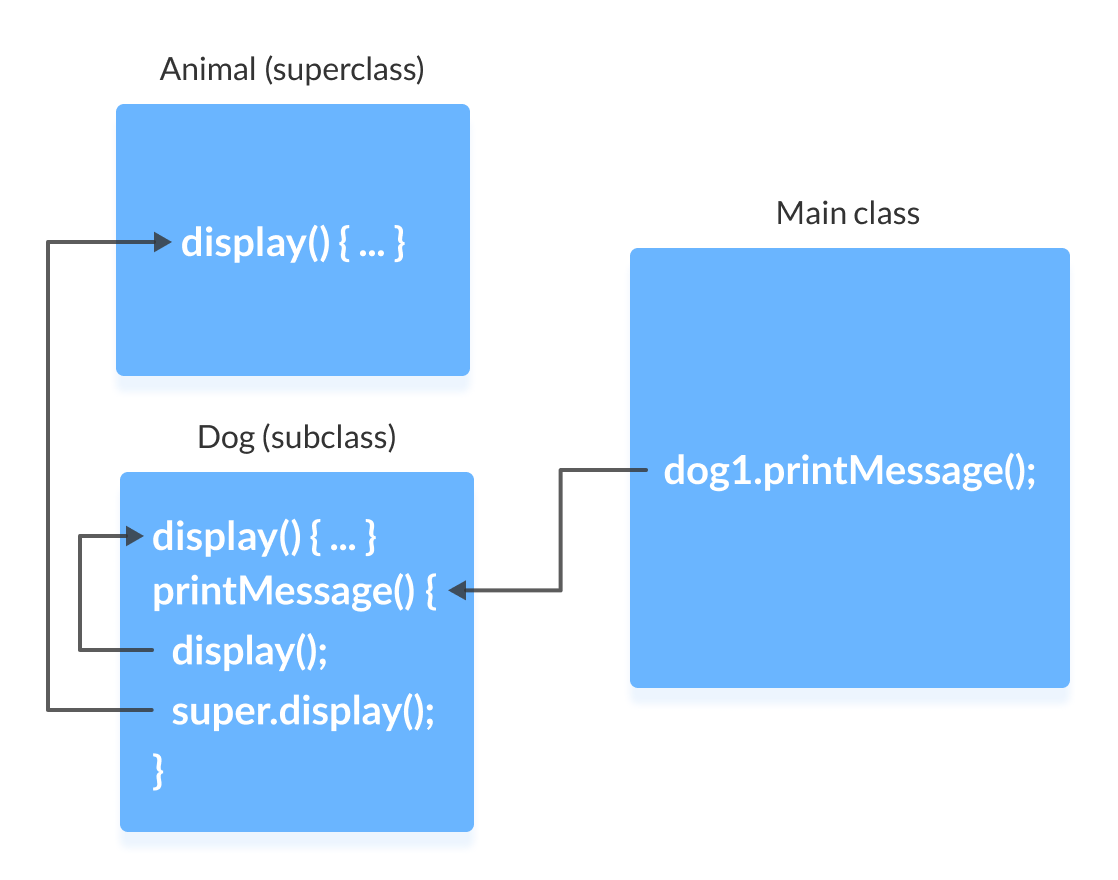
[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**

I am a dog

I am an animal

Here, how the above program works.



**2. Access Attributes of the Superclass**

The superclass and subclass can have attributes with the same name. We use the super keyword to access the attribute of the superclass.

**Example 3: Access superclass attribute**

class Animal {

protected String type="animal";

}

class Dog extends Animal {

public String type="mammal";

public void printType() {

System.out.println("I am a " + type);

System.out.println("I am an " + super.type);

}

}

class Main {

public static void main(String[] args) {

Dog dog1 = new Dog();

dog1.printType();

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**:

I am a mammal

I am an animal

In this example, we have defined the same instance field type in both the superclass Animal and the subclass Dog.

We then created an object dog1 of the Dog class. Then, the printType() method is called using this object.

Inside the printType() function,

* type refers to the attribute of the subclass Dog.
* super.type refers to the attribute of the superclass Animal.

Hence, System.out.println("I am a " + type); prints I am a mammal. And, System.out.println("I am an " + super.type); prints I am an animal.

**3. Use of super() to access superclass constructor**

As we know, when an object of a class is created, its default constructor is automatically called.

To explicitly call the superclass constructor from the subclass constructor, we use super(). It's a special form of the super keyword.

super() can be used only inside the subclass constructor and must be the first statement.

**Example 4: Use of super()**

class Animal {

// default or no-arg constructor of class Animal

Animal() {

System.out.println("I am an animal");

}

}

class Dog extends Animal {

// default or no-arg constructor of class Dog

Dog() {

// calling default constructor of the superclass

super();

System.out.println("I am a dog");

}

}

class Main {

public static void main(String[] args) {

Dog dog1 = new Dog();

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

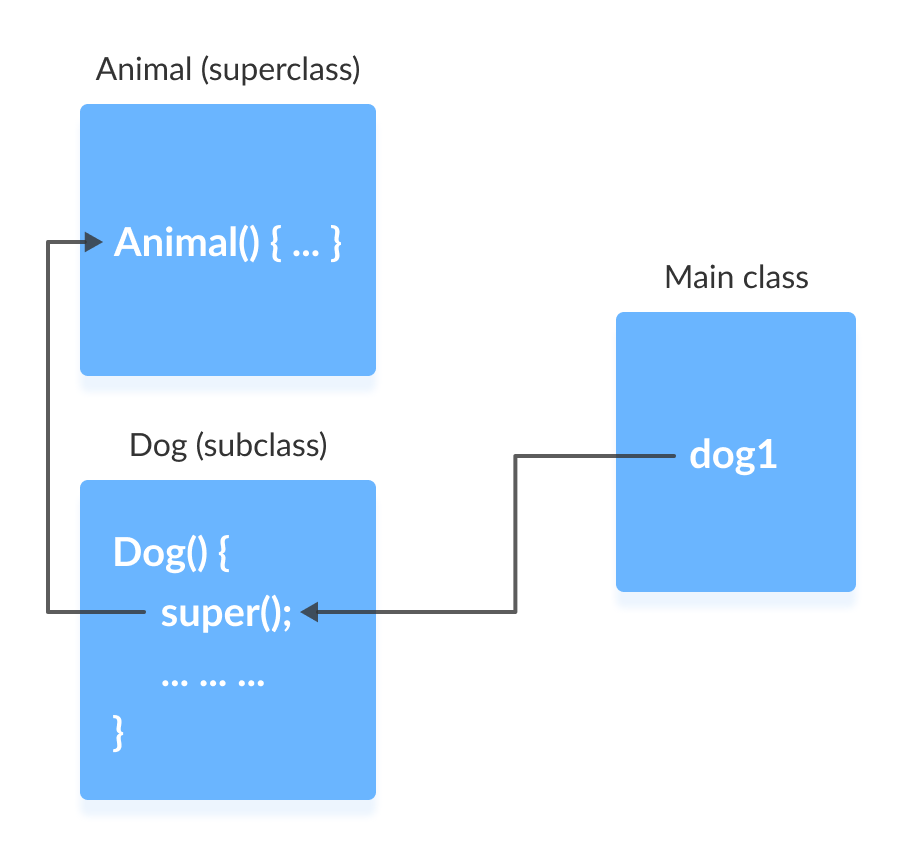
**Output**

I am an animal

I am a dog

Here, when an object dog1 of Dog class is created, it automatically calls the default or no-arg constructor of that class.

Inside the subclass constructor, the super() statement calls the constructor of the superclass and executes the statements inside it. Hence, we get the output I am an animal.



The flow of the program then returns back to the subclass constructor and executes the remaining statements. Thus, I am a dog will be printed.

However, using super() is not compulsory. Even if super() is not used in the subclass constructor, the compiler implicitly calls the default constructor of the superclass.

**So, why use redundant code if the compiler automatically invokes super()?**

It is required if the **parameterized constructor (a constructor that takes arguments)** of the superclass has to be called from the subclass constructor.

The parameterized super() must always be the first statement in the body of the constructor of the subclass, otherwise, we get a compilation error.

**Example 5: Call Parameterized Constructor Using super()**

class Animal {

// default or no-arg constructor

Animal() {

System.out.println("I am an animal");

}

// parameterized constructor

Animal(String type) {

System.out.println("Type: "+type);

}

}

class Dog extends Animal {

// default constructor

Dog() {

// calling parameterized constructor of the superclass

super("Animal");

System.out.println("I am a dog");

}

}

class Main {

public static void main(String[] args) {

Dog dog1 = new Dog();

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

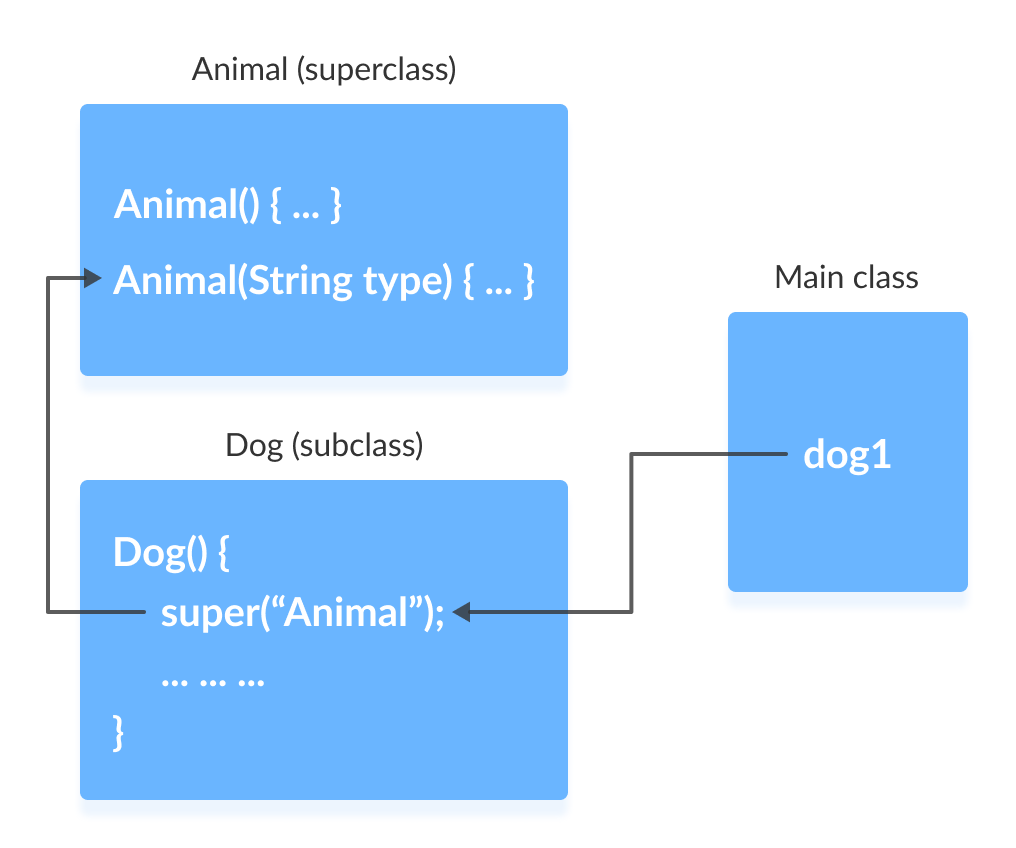
**Output**

Type: Animal

I am a dog

The compiler can automatically call the no-arg constructor. However, it cannot call parameterized constructors.

If a parameterized constructor has to be called, we need to explicitly define it in the subclass constructor.



Note that in the above example, we explicitly called the parameterized constructor super("Animal"). The compiler does not call the default constructor of the superclass in this case.