

# Super Keyword

In this lesson, you'll get to know about the uses of the super keyword in Java.

## WE'LL COVER THE FOLLOWING ^

- What is the `super` Keyword?
- Use Cases of the `super` Keyword
  - Accessing Parent Class Fields
  - Calling a Parent Class Method
  - Using with Constructors

## What is the `super` Keyword? #

As you already know that this keyword in Java is used to refer to the *instance* of the current class.

In a similar fashion, the `super` keyword in Java is used to refer to the *SuperClass* members from inside the immediate *Subclass*. The use of `super` comes into play when we implement inheritance.

## Use Cases of the `super` Keyword #

The super keyword is used in *three* major contexts:

### Accessing Parent Class Fields #

Consider the fields named as `fuelCap` defined inside a `Vehicle` class to keep track of the *fuel capacity* of a vehicle. Another class named as `Car` extends from this `Vehicle` class. We declare a field inside the `Car` class with the same name i.e. `fuelCap` but different value. Now if we want to refer to the `fuelCap` field of the *SuperClass* inside the *Subclass*, we will then have to use the `super` keyword.

Let's understand this using a bit of code.

```

class Vehicle { //Base class vehicle

    int fuelCap = 90; //fuelCap field inside SuperClass
}

class Car extends Vehicle { // sub class Car extending from Vehicle

    int fuelCap = 50; //fuelCap field inside SubClass

    public void display() {
        //accessing the field of parent class using super*/
        System.out.println("Fuel Capacity from the Vehicle class: " + super.fuelCap);
        //without using super the field of current class shadows the field of parant class*/
        System.out.println("Fuel Capacity from the Car class: " + fuelCap);
    }
}

class Main {

    public static void main(String[] args) {
        Car corolla = new Car();
        corolla.display();
    }
}

```



## Calling a Parent Class Method #

Just like the fields, **super** is also used with the methods. Whenever a *SuperClass* and the immediate *SubClass* have any methods with the **same name** we use **super** to access the methods from the *SuperClass* inside the *SubClass*. Let's go through an example:

```

class Vehicle {          //Base class vehicle

    public void display() { //display method inside SuperClass
        System.out.println("I am from the Vehicle Class");
    }
}

class Car extends Vehicle { // sub class Car extending from Vehicle

    public void display() { //display method inside SubClass
        System.out.println("I am from the Car Class");
    }
}

```



```

}

public void printOut(){

    System.out.println("The display() call with super:");
    super.display(); //calling the display() of Vehicle(SuperClass)
    System.out.println("The display() call without super:");
    display();       //calling the display() of the Car(SubClass)
}

}

class Main {

    public static void main(String[] args) {
        Car corolla = new Car();
        corolla.printOut();
    }

}

```



## Using with Constructors #

Another very important use of the keyword `super` is to call the *constructor* of the *SuperClass* from inside of the *constructor* of the *SubClass*.

**Important Note:** When you create an Object of a *SubClass* type at the same time, an Object of *SuperClass* type is created by calling implicitly the constructor of *SuperClass*.

The syntax of the constructor call is as follows:

```

super(); //calls the (no argument) constructor if a no argument construct
or is defined in the SuperClass

super(parameters); //calls the parameterized constructor of the SuperClas
s with matching parameters from the SubClass constructor

```

The above two lines are the generalized syntax for the *SuperClass* constructor call.

**Very Important:** The call to the SuperClass constructor using `super()` should always be the first line of code inside the constructor of the

## SubClass.

Let's look at an example of a constructor calling using `super()`.

**Note:** The below code will give an error as there is no call to the SuperClass constructor from inside of the SubClass constructor.

```
class Vehicle {                                //base class of vehicle

    private String make;    //
    private String color;   // Vehicle Fields
    private int year;       //
    private String model;   //

    public Vehicle(String make, String color, int year, String model) {
        this.make = make;    //
        this.color = color;  // Constructor of Vehicle
        this.year = year;    //
        this.model = model;  //
    }

    public void printDetails() { //public method to print details
        System.out.println("Manufacturer: " + make);
        System.out.println("Color: " + color);
        System.out.println("Year: " + year);
        System.out.println("Model: " + model);
    }
}

class Car extends Vehicle {    //derived class of Car

    private String bodyStyle; //Car field

    public Car(String make, String color, int year, String model, String bodyStyle) {
        //super(make, color, year, model); //parent class constructor
        this.bodyStyle = bodyStyle;
    }

    public void carDetails() { //details of car
        printDetails();        //calling method from parent class
        System.out.println("Body Style: " + bodyStyle);
    }
}

class Main {

    public static void main(String[] args) {
        Car elantraSedan = new Car("Hyundai", "Red", 2019, "Elantra", "Sedan"); //creation of
        elantraSedan.carDetails(); //calling method to print details
    }
}
```

Now let's uncomment the above highlighted line in the code widget and try running the code again. It will execute this time.

```
class Vehicle {                                //base class of vehicle

    private String make;    //
    private String color;   // Vehicle Fields
    private int year;       //
    private String model;   //

    public Vehicle(String make, String color, int year, String model) {
        this.make = make;    //
        this.color = color;  // Constructor of Vehicle
        this.year = year;    //
        this.model = model;  //
    }

    public void printDetails() { //public method to print details
        System.out.println("Manufacturer: " + make);
        System.out.println("Color: " + color);
        System.out.println("Year: " + year);
        System.out.println("Model: " + model);
    }
}

class Car extends Vehicle {    //derived class of Car

    private String bodyStyle; //Car field

    public Car(String make, String color, int year, String model, String bodyStyle) {
        super(make, color, year, model); //parent class constructor
        this.bodyStyle = bodyStyle;
    }

    public void carDetails() { //details of car
        printDetails();        //calling method from parent class
        System.out.println("Body Style: " + bodyStyle);
    }
}

class Main {

    public static void main(String[] args) {
        Car elantraSedan = new Car("Hyundai", "Red", 2019, "Elantra", "Sedan"); //creation of
        elantraSedan.carDetails(); //calling method to print details
    }
}
```



This time the execution is successful.

**Note:** In a constructor we can include a call to `super()` **or** `this()` but not both. Also, these calls can only be used inside the *constructors*.

---

So this was pretty much about the super keyword. In the next lesson, we will discuss the different types of inheritance.