Super keyword

- Super keyword refers to the objects of super class,it is used when we want to call. The super class variable ,method & constructor through sub class object.
- Whenever the super class & sub class variable and method name both are same than it can used only.
- To avoid the confusion between super class and sub classes variables and method that have same name we should use super keyword.

Variable:

- We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields.

Program:

```
class Animal {
    String color = "white";
}

class Dog extends Animal {
    String color = "black";

    void printColor() {
        System.out.println(color);// prints color of Dog class
        System.out.println(super.color);// prints color of Animal class
    }
}

class TestSuper1 {
    public static void main(String args[]) {
        Dog d = new Dog();
        d.printColor();
    }
}
OUTPUT:
black
white
```

method:

- The super keyword can also be used to invoke parent class method. It should be used if subclass contains the same method as parent class. In other words, it is used if method is overridden.

Program:

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

class Dog extends Animal {
    void eat() {
```

```
        System.out.println("eating bread...");
    }

    void bark() {
        System.out.println("barking...");
    }

    void work() {
        super.eat();
        bark();
    }
}

class TestSuper2 {
    public static void main(String args[]) {
        Dog d = new Dog();
        d.work();
    }
}
OUTPUT:- eating...
barking...
```

Constructor

- The super keyword can also be used to invoke the parent class constructor. Let's see a simple example:

Program:

```
    class Animal{
        Animal(){System.out.println("animal is created");}
        }
    class Dog extends Animal{
        Dog(){
        super();
        System.out.println("dog is created");
        }
        }
        class TestSuper3{
        public static void main(String args[]){
        Dog d=new Dog();
        }}
OUTPut:-
animal is created
dog is created
```

super example: real use

Let's see the real use of super keyword. Here, Emp class inherits Person class so all the properties of Person will be inherited to Emp by default. To initialize all the property, we are using parent class constructor from child class. In such way, we are reusing the parent class constructor.

```
class Person {
    int id;
    String name;

    Person(int id, String name) {
        this.id = id;
        this.name = name;
```

```
    }
}

class Emp extends Person {
    float salary;

    Emp(int id, String name, float salary) {
        super(id, name);// reusing parent constructor
        this.salary = salary;
    }

    void display() {
        System.out.println(id + " " + name + " " + salary);
    }
}

class TestSuper5 {
    public static void main(String[] args) {
        Emp e1 = new Emp(1, "ankit", 45000f);
        e1.display();
    }
}
OUTPUT:- 1 ankit 45000.0
```

This keyword

- This keyword refers to the current object inside a method or constructor

1) this: to refer current class instance variable

- The this keyword can be used to refer current class instance variable. If there is ambiguity between the instance variables and parameters, this keyword resolves the problem of ambiguity

Program:

```
class Student {
    int rollno;
    String name;
    float fee;

    Student(int rollno, String name, float fee) {
        this.rollno = rollno;
        this.name = name;
        this.fee = fee;
    }

    void display() {
        System.out.println(rollno + " " + name + " " + fee);
    }
}

class TestThis2 {
    public static void main(String args[]) {
        Student s1 = new Student(111, "ankit", 5000f);
        Student s2 = new Student(112, "sumit", 6000f);
        s1.display();
        s2.display();
    }
}
OUTPUT:-
111 ankit 5000.0
112 sumit 6000.0
```

2) this: to invoke current class method

- You may invoke the method of the current class by using the this keyword. If you don't use the this keyword, compiler automatically adds this keyword while invoking the method. Let's see the example

Program:

```
class A {
    void m() {
        System.out.println("hello m");
    }

    void n() {
        System.out.println("hello n");
        // m();//same as this.m()
        this.m();
    }
}

class TestThis4 {
    public static void main(String args[]) {
        A a = new A();
        a.n();
    }
}
Output:-
hello n
hello m
```

3) this() : to invoke current class constructor

- The this() constructor call can be used to invoke the current class constructor. It is used to reuse the constructor. In other words, it is used for constructor chaining.

Program:

```
class Student {
    int rollno;
    String name, course;
    float fee;

    Student(int rollno, String name, String course) {
        this.rollno = rollno;
        this.name = name;
        this.course = course;
    }

    Student(int rollno, String name, String course, float fee) {
        this(rollno, name, course);// reusing constructor
        this.fee = fee;
    }

    void display() {
        System.out.println(rollno + " " + name + " " + course + " " + fee);
    }
}

class TestThis7 {
    public static void main(String args[]) {
```

```
        Student s1 = new Student(111, "ankit", "java");
        Student s2 = new Student(112, "sumit", "java", 6000f);
        s1.display();
        s2.display();
    }
}
Output-
111 ankit java 0.0
112 sumit java 6000.0
```

Static block	instance block
Known only as **static initialization block** in java.	Also known as **non-static initialization block** in java.
static blocks executes **before instance blocks** in java.	**instance blocks** executes **after static blocks** in java.
Only static variables can be **accessed** inside **static block**	Static and non-static variables (instance variables) can be **accessed** inside **instance block.**
static blocks can be used for **initializing** static **variables** **or** calling any static **method** in java.	**instance blocks** can be used for initializing instance **variables** ***or*** calling any instance **method** in java.
static blocks executes **when class is loaded** in java.	**instance block** executes only **when instance of class is created**, not called when class is loaded in java.
this keyword cannot be used in **static blocks**.	**this keyword** can be used in **instance block**.

Access Modifiers in Java

- **The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.**

There are four types of Java access modifiers:

1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.

2. **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y