1) super is used to refer immediate parent class instance 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.

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

class Dog extends Animal
{
String color="black";
        void printColor()
        {
    System.out.println(color);//prints colo
  r 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
```

Another Example from Text book

This second form of **super** is most applicable to situations in which member names of a subclass hide members by the same name in the superclass. Consider this simple class hierarchy:

```
// Using super to overcome name hiding.
class A {
int i;
}

// Create a subclass by extending class A.
class B extends A {
int i; // this i hides the i in A
B(int a, int b) {
super.i = a; // i in A
i = b; // i in B
}
void show() {
System.out.println("i in superclass: " + super.i);
System.out.println("i in subclass: " + i);
}
}
class UseSuper {
public static void main(String args[]) {
B subOb = new B(1, 2);
subOb.show();
}
}
```

This program displays the following:

```
i in superclass: 1
i in subclass: 2
```

2) super can be used to invoke parent class 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.

```
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...
```

3) super is used to invoke parent class constructor.

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

```
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();
        }
}
```

o/p

```
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
```

since **super( )** must be the
first statement executed in a subclass' constructor, this order is the same whether or not **super( )**

is used. If **super( )** is not used, then the default or parameterless constructor of each superclass
will be executed. The following program illustrates when constructors are executed:

```
// Demonstrate when constructors are called.
// Create a super class.
class A {
A() {
System.out.println("Inside A's constructor.");
}
}
// Create a subclass by extending class A.
class B extends A {
B() {
System.out.println("Inside B's constructor.");
}
}
// Create another subclass by extending B.
class C extends B {
C() {
System.out.println("Inside C's constructor.");
}
}
class CallingCons {
public static void main(String args[]) {
C c = new C();
}
}
```

The output from this program is shown here:

```
Inside A’s constructor
Inside B’s constructor
Inside C’s constructor
```