**super keyword**

The **super** is a reference variable that is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly i.e. referred by super reference variable.

**Usage of super Keyword**

1. super is used to refer immediate parent class instance variable.
2. super() is used to invoke immediate parent class constructor.
3. super is used to invoke immediate parent class method.

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

***Problem without super keyword***

1. class Vehicle{
2. int speed=50;
3. }
4.
5. class Bike extends Vehicle{
6. int speed=100;
7.
8. void display(){
9. System.out.println(speed);//will print speed of Bike
10. }
11. public static void main(String args[]){
12. Bike b=new Bike();
13. b.display();
14.
15. }
16. }

Output:100

|  |
| --- |
| In the above example Vehicle and Bike both class have a common property speed. Instance variable of current class is refered by instance bydefault, but I have to refer parent class instance variable that is why we use super keyword to distinguish between parent class instance variable and current class instance variable.  |

***Solution by super keyword***

1. //example of super keyword
2.
3. class Vehicle{
4. int speed=50;
5. }
6.
7. class Bike extends Vehicle{
8. int speed=100;
9.
10. void display(){
11. System.out.println(super.speed);//will print speed of Vehicle now
12. }
13. public static void main(String args[]){
14. Bike b=new Bike();
15. b.display();
16.
17. }
18. }

Output:50

**2) super is used to invoke parent class constructor.**

|  |
| --- |
| The super keyword can also be used to invoke the parent class constructor as given below: |

1. class Vehicle{
2. Vehicle(){System.out.println("Vehicle is created");}
3. }
4.
5. class Bike extends Vehicle{
6. Bike(){
7. super();//will invoke parent class constructor
8. System.out.println("Bike is created");
9. }
10. public static void main(String args[]){
11. Bike b=new Bike();
12.
13. }
14. }

Output:Vehicle is created

 Bike is created

**super() is added in each class constructor automatically by compiler.**



|  |
| --- |
| As we know well that default constructor is provided by compiler automatically but it also adds super() for the first statement.If you are creating your own constructor and you don't have either this() or super() as the first statement, compiler will provide super() as the first statement of the constructor.  |

**Another example of super keyword where super() is provided by the compiler implicitly.**

1. class Vehicle{
2. Vehicle(){System.out.println("Vehicle is created");}
3. }
4.
5. class Bike extends Vehicle{
6. int speed;
7. Bike(int speed){
8. this.speed=speed;
9. System.out.println(speed);
10. }
11. public static void main(String args[]){
12. Bike b=new Bike(10);
13. }
14. }

Output:Vehicle is created

 10

**3) 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 in case subclass contains the same method as parent class as in the example given below: |

1. class Person{
2. void message(){System.out.println("welcome");}
3. }
4.
5. class Student extends Person{
6. void message(){System.out.println("welcome to java");}
7.
8. void display(){
9. message();//will invoke current class message() method
10. super.message();//will invoke parent class message() method
11. }
12.
13. public static void main(String args[]){
14. Student s=new Student();
15. s.display();
16. }
17. }

Output:welcome to java

 welcome

|  |
| --- |
| In the above example Student and Person both classes have message() method if we call message() method from Student class, it will call the message() method of Student class not of Person class because priority is given to local.  |

|  |
| --- |
| In case there is no method in subclass as parent, there is no need to use super. In the example given below message() method is invoked from Student class but Student class does not have message() method, so you can directly call message() method.  |

**Program in case super is not required**

1. class Person{
2. void message(){System.out.println("welcome");}
3. }
4.
5. class Student extends Person{
6.
7. void display(){
8. message();//will invoke parent class message() method
9. }
10.
11. public static void main(String args[]){
12. Student s=new Student();
13. s.display();
14. }
15. }

Output:welcome

## Using the super keyword:

When invoking a superclass version of an overridden method the **super** keyword is used.

class Animal{

 public void move(){

 System.out.println("Animals can move");

 }

}

class Dog extends Animal{

 public void move(){

 super.move(); // invokes the super class method

 System.out.println("Dogs can walk and run");

 }

}

public class TestDog{

 public static void main(String args[]){

 Animal b = new Dog(); // Animal reference but Dog object

 b.move(); //Runs the method in Dog class

 }

}

This would produce the following result:

Animals can move

Dogs can walk and run

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

Any Java object that can pass more than one IS-A test is considered to be polymorphic. In Java, all Java objects are polymorphic since any object will pass the IS-A test for their own type and for the class Object.

It is important to know that the only possible way to access an object is through a reference variable. A reference variable can be of only one type. Once declared, the type of a reference variable cannot be changed.

The reference variable can be reassigned to other objects provided that it is not declared final. The type of the reference variable would determine the methods that it can invoke on the object.

A reference variable can refer to any object of its declared type or any subtype of its declared type. A reference variable can be declared as a class or interface type.

## Example:

Let us look at an example.

public interface Vegetarian{}

public class Animal{}

public class Deer extends Animal implements Vegetarian{}

Now, the Deer class is considered to be polymorphic since this has multiple inheritance. Following are true for the above example:

* A Deer IS-A Animal
* A Deer IS-A Vegetarian
* A Deer IS-A Deer
* A Deer IS-A Object

When we apply the reference variable facts to a Deer object reference, the following declarations are legal:

Deer d = new Deer();

Animal a = d;

Vegetarian v = d;

Object o = d;

All the reference variables d,a,v,o refer to the same Deer object in the heap.