Constructor inheritance and use of Super keyword to access superclass constructor

Class can not only be determined by state and behavior of class but also by each of its superclasses. Due to that reason it is not sufficient to execute a constructor of that class but also need to initialize each constructor of superclasses.

A superclass constructor must execute before a subclass constructor. So that the state and behavior defined by the superclass may be correctly and completely initialized before a subclass constructor executes.

Java compiler assumes that the first line of every constructor is an implicit call to the default superclass constructor unless you explicitly use super() or this().

Note that Super keyword is used to explicitly invoke a superclass constructor. If you use this form, it must appear as the first statement of the constructor to ensure that the superclass constructor executes before the subclass constructor in java.

Syntax to call Superclass Constructor: super(args);

args is an optional list of arguments of the superclass constructor.

Note that you cannot use both **super()** and **this()** at the same time in same constructor because **both must appear as the first statement of the constructor.**

EXAMPLE OF CONSTRUCTOR INHERITANCE AND USE OF SUPER KEYWORD TO CALL CONSTRUCTOR OF SUPER CLASS

Example-1: Program that shows the default execution of constructor when classes are inherited

```
class Person{
    Person()    {
            System.out.println("Constructor of person class");
     }
}

class Employee extends Person{
    Employee()    {
            System.out.println("Constructor of employee class");
     }
}

class PermanentEmployee extends Employee{
    PermanentEmployee()    {
            System.out.println("Constructor of permanent employee class");
     }
}

class ConstructorInheritanceDemo{
     public static void main(String args[]) {
            PermanentEmployee pObj = new PermanentEmployee();
      }
}
```

Output

Constructor of person class
Constructor of employee class
Constructor of permanent employee class

Note that the superclass person and Employee constructors executes before the subclass PermanentEmployee.

Example 2 : Program that illustrates the explicit use of super() in an inheritance hierarchy.

```
class Person{
    String FirstName;
    String LastName;

    Person(String fName, String lName){
            FirstName = fName;
            LastName = lName;
     }
}

class Student extends Person{
    int id;
    String standard;
    String instructor;

    Student(String fName, String lName, int nId, String stnd, String instr){
        super(fName,lName);
        id = nId;
        standard = stnd;
        instructor = instr;
     }
}

class SuperKeywordForConstructorDemo{
      public static void main(String args[]){
       Student sObj = new Student("Jacob","Smith",1,"1 - B","Roma");

            System.out.println("Student : ");
            System.out.println("First Name : " + sObj.FirstName);
            System.out.println("Last Name : " + sObj.LastName);
            System.out.println("ID : " + sObj.id);
            System.out.println("Standard : " + sObj.standard);
            System.out.println("Instructor : " + sObj.instructor);
       }
}
```

Output

Student :
First Name : Jacob
Last Name : Smith
ID : 1
Standard : 1 - B
Instructor : Roma

In Java, constructor of base class with no argument gets automatically called in derived class constructor. For example, output of following program is:
Base Class Constructor Called
Derived Class Constructor Called

```
// filename: Main.java
class Base {
  Base() {
    System.out.println("Base Class Constructor Called ");
  }
}

class Derived extends Base {
  Derived() {
    System.out.println("Derived Class Constructor Called ");
  }
}

public class Main {
  public static void main(String[] args) {
    Derived d = new Derived();
  }
}
```

But, if we want to call parameterized contructor of base class, then we can call it using super(). The point to note is **base class comstructor call must be the first line in derived class constructor**. For example, in the following program, super(_x) is first line derived class constructor.

```
// filename: Main.java
class Base {
  int x;
  Base(int _x) {
    x = _x;
  }
}

class Derived extends Base {
  int y;
  Derived(int _x, int _y) {
    super(_x);
    y = _y;
  }
  void Display() {
    System.out.println("x = "+x+", y = "+y);
  }
}

public class Main {
  public static void main(String[] args) {
    Derived d = new Derived(10, 20);
    d.Display();
  }
}
```