



FIGURE 7.2

We can refer to any member of the current object from a method or a constructor by using **this**. In java, **this** keyword is used in many cases. Let us consider some major uses of **this** keyword :

- (i) **The this keyword can be used to refer current class instance variable.**
 If name of parameter (format arguments) or local variables and instance variables are same, then the parameter hides the instance variable and we get wrong result.

Example 7.17

Study the following program.

```

class Student
{
    int roll;
    int marks;
    Student (int roll, int marks)
    {
        roll = roll;
        marks = marks;
    }
    void display()
    {
        System.out.println(roll + " " + marks);
    }
    public static void main(String args[])
    {
        Student s1 = new Student(1001, 80);
        Student s2 = new Student(1002, 90);
        s1.display();
        s2.display();
    }
}
  
```

Output :

```

0 0
0 0
  
```

In the above example, name of parameter (**formal arguments**) and **instance variables** are same, therefore, we get wrong result. Solution of this problem are :

- (a) Use different names for parameters or local variables and instance variables.
- (b) Use **this** keyword to distinguish between parameters or local variables and instance variables.

Example 7.18

Solution of the above problem by this keyword

```
//example of this keyword
class Student
{
    int roll;
    int marks;
    Student (int roll, int marks)
    {
        ✓ this.roll = roll;
        ✓ this.marks = marks;
    }

    void display()
    {
        System.out.println(roll + " " + marks);
    }

    public static void main(String args[])
    {
        Student s1 = new Student(1001, 80);
        Student s2 = new Student(1002, 90);
        s1.display();
        s2.display();
    }
}
```

Output :

```
1001 80
1002 90
```


- This function - This()*
- (ii) **This()** can be used to invoke current class constructor.
- From within a constructor, we can use **this** keyword to call another constructor of the same class. Calling a constructor from another constructor of the same class is also called as **constructor chaining**. This approach is useful if you have many constructors in the class and want to reuse that constructor.

Example 7.19

// Program of this () constructor call (constructor chaining)

```
class Student
{
    int roll;
    int marks;
    Student()           //Constructor 1
    {
        System.out.println("Default Constructor is invoked");
    }

    Student (int roll, int marks)    // Constructor 2
    {
        this(); //Use to invoke current class constructor i.e. constructor 1
        this.roll = roll;
        this.marks = marks;
    }

    void display()
    {
        System.out.println(roll + " " + marks);
    }

    public static void main(String args[])
    {
        Student s1 = new Student(1001, 80);
        Student s2 = new Student(1002, 90);
        s1.display();
        s2.display();
    }
}
```

Output :

```
Default Constructor is invoked
Default Constructor is invoked
1001    80
1102    90
```

Note that the **this()** constructor call should be used to reuse the constructor in the constructor.

7.15 Static Members

There are cases when we want to define a class member to be used independently of any object of that class. To create such a member, its declaration is preceded with the keyword **static**. **When a member is declared static, it can be accessed before any objects of its class are created, and without reference to any object.**

The static can be :

- (i) variable (also known as class variable)
- (ii) method (also known as class method)
- (iii) block



Note :

Advantage of Static variable is that It makes your program memory efficient. It saves memory.

7.15.1 Static Variable

If you declare any variable as static, it is known static variable.

Properties of static variables are :

- The static variable can be used to refer the common property of a classes. For example, company name of employee, college name of students etc.
- The static variable gets memory only once.

Let us understanding advantage of static variable

Example 7.24

Consider the following class

```
class Student
{
    int rollno;           // instance variable
    String name;          // instance variable
    String college="D.S. College"; //instance variable
}
```

Suppose there are 3500 students in my college, now all instance variables will get memory each time when object is created. All student have its unique **rollno** and **name**. But, all students have same college name. If we make it static, this field will get memory only once. Therefore, we can saves memory using static variable.

Example 7.25

Study the following program

```
class Student
```

```
{  
    int rollno;  
    String name;  
    static String college = "D.S. College";
```

```
    Student(int r, String n)
```

```
{  
        rollno = r;  
        name = n;  
    }
```

```
    void display ()
```

```
{  
        System.out.println(rollno + " " + name + " " + college);  
    }
```

```
    public static void main(String args[ ])
```

```
{  
        Student s1 = new Student(1001, "Shivank");  
        Student s2 = new Student(1002, "Shitiz");  
        s1.display();  
        s2.display();  
    }  
}
```

Output :

```
1001 Shivank D.S. College  
1002 Shitiz   D.S. College
```

Instance variable declared as **static** are known as **static variables**, **global variables** or **class variables**. One common **use** of **static** variable is to count the number of objects created in a class.

[201]

7.15.2 Static Method

If you apply static keyword with any method, it is known as static method or class method.

Properties of static methods are :

- A static method belongs to the class rather than object of a class.
- A static method can be invoked without the need for creating an instance of a class (or object of a class).
- Static method can access static data member and can change the value of it.

[202]



Note :

main() method is also declared as **static** because

- (a) Object is not required to call static method.
- (b) It must be called before any objects exist.

Methods that are of general utility but do not directly affect an instance of that class are usually declared as class methods. Java class libraries contain a large number of class methods. For example, the **Math** class of Java library defines many static methods to perform math operations that can be used in any program.

Example 7.27

Program to calculate the cube of a given number by static method.

```
class calcube
{
    static int cube(int x)
    {
        return x*x*x;
    }

    public static void main(String args[])
    {
        int result=calcube.cube (6);
        System.out.println(result);
    }
}
```

Output :

216

Restrictions for Static Method

There are three restrictions for the static method.

- The static method can not use non static data member.
- The static method can only call other static methods.
- The **this** and **super** keywords cannot be used in static context.

7.3 Static Block

We can declare a **static block** which gets executed exactly once, when the class is first loaded.

Properties of static block are :

- It is used to initialize the static data member.
- It is executed before main method at the time of classloading.

Example 7.28

Consider the following program to illustrates the use of static block.

```
class Sblock
{
    static
    {
        System.out.println("Static block is invoked.");
    }

    public static void main(String args[])
    {
        System.out.println("We are in Java main method.");
    }
}
```

Output :

Static block is invoked.

We are in Java main method.