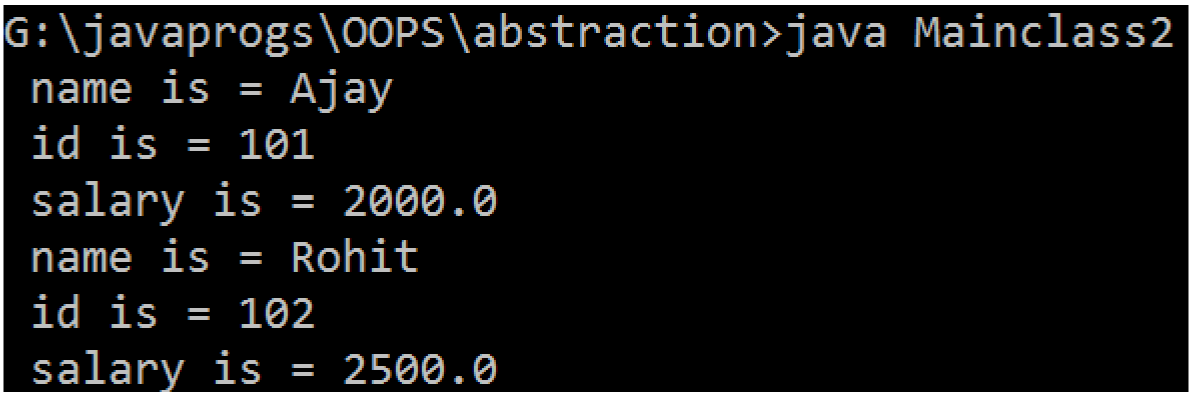
```
class ParameterizedConstructor
{
	public static void main(String[] args)
	{
		Sample s = new Sample("Ajay", 101, 2000);
		s.display();
		Sample s2 = new Sample("Rohit", 102, 2500);
		s2.display();

	}
}
```



```
G:\javaprogs\OOPS\abstraction>java Mainclass2
 name is = Ajay
 id is = 101
 salary is = 2000.0
 name is = Rohit
 id is = 102
 salary is = 2500.0
```

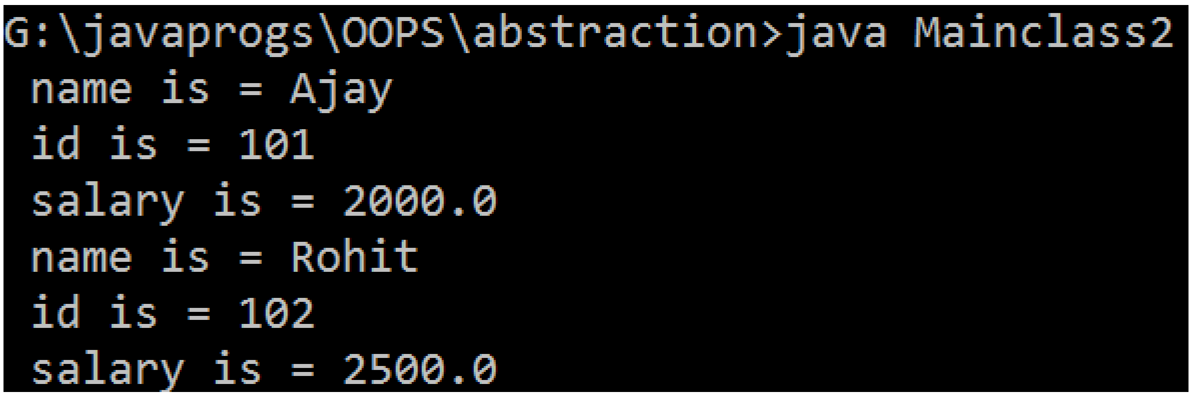
CHAPTER 4 : THIS KEYWORD AND THIS STATEMENT()

This keyword

- It is used to differentiate between local and global variables whenever they have same names.

```
class Sample
{
	String name;
	int id;
	double sal;

	public Sample(String name, int id, double sal)
	{
//		System.out.println("This is sample const...");
		this.name=name;
		this.id=id;
```

```
                this.sal=sal;
                return;
        }
        public void display()
        {
                System.out.println(" name is = " + name);;
                System.out.println(" id is = " + id);;
                System.out.println(" salary is = " + sal);;
        }}
class ThisOperator
{
        public static void main(String[] args)
        {
                Sample s = new Sample("Ajay", 101, 2000);
                s.display();
                Sample s2 = new Sample("Rohit", 102, 2500);
                s2.display();
        }}
```

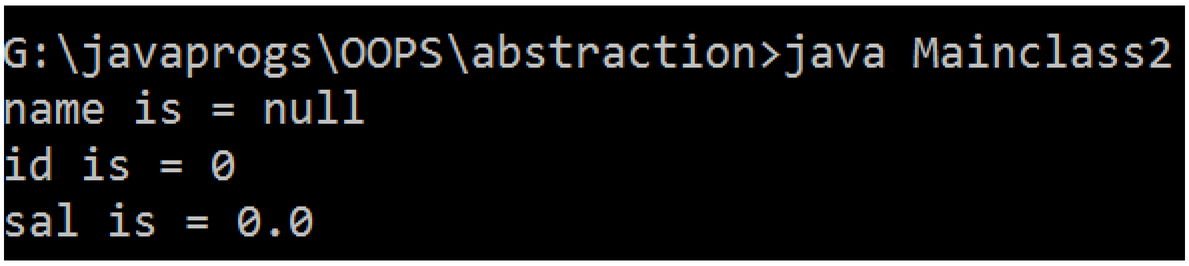


```
G:\javaprogs\OOPS\abstraction>java Mainclass2
 name is = Ajay
 id is = 101
 salary is = 2000.0
 name is = Rohit
 id is = 102
 salary is = 2500.0
```

- This keyword is a special type of reference variables which always points to active or current instance of the class.
- This keyword can not be used within the static method.
- This keyword can be used only within the non-static methods and constructors of the class.

// we can not like do this, it print the default values of global variables.

```
class Demoa
{
	String name;
	int id;
	double sal;

	public Demoa(String name, int id, double sal)
	{
		name=name;
		id=id;
		sal=sal;
	}
	public void disp()
	{
		System.out.println("name is = "+this.name);
		System.out.println("id is = "+this.id);
		System.out.println("sal is = "+this.sal);
	}
	public static void main(String[] args)
	{
		Demoa a = new Demoa("ajay", 101, 1020.05);
		a.disp();
	}
}
```

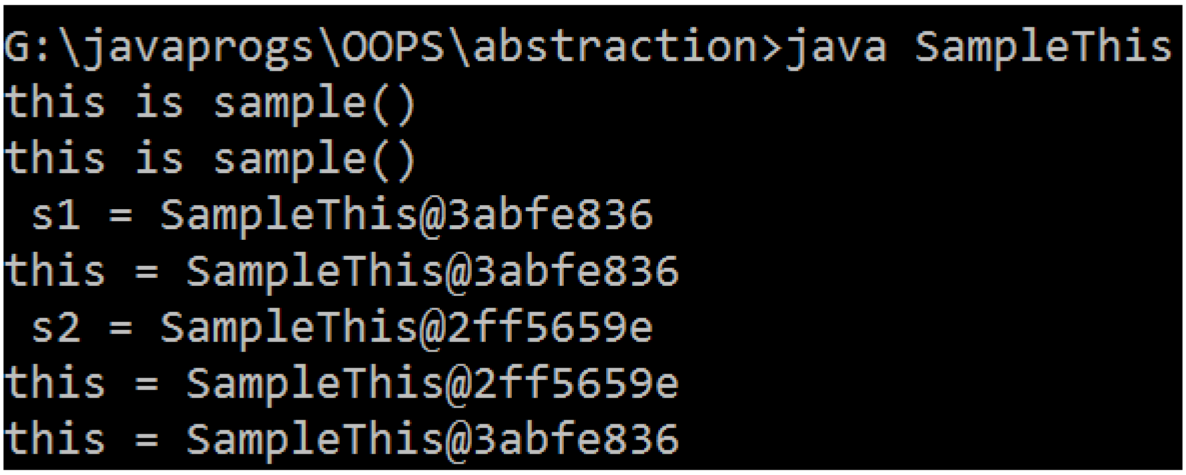


```
G:\javaprogs\OOPS\abstraction>java Mainclass2
name is = null
id is = 0
sal is = 0.0
```

Program

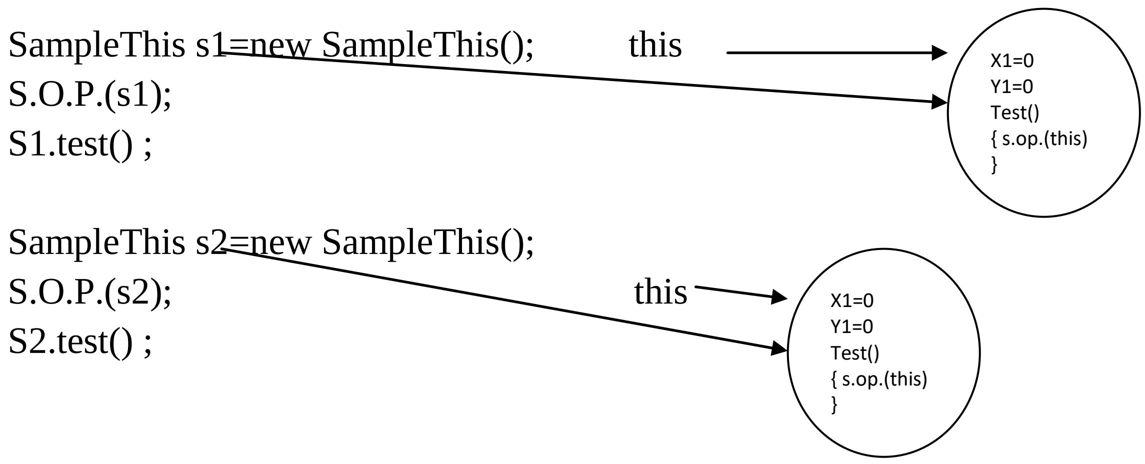
//use of this keyword

```
class SampleThis
{
        int x1;
        double y1;

        public SampleThis()
        {
                System.out.println("this is sample()");
        }
        public void test()
        {
                System.out.println("this = "+this);
        }

public static void main(String[] args) {
        SampleThis s1=new SampleThis();
        SampleThis s2=new SampleThis();
        System.out.println(" s1 = "+s1);
        s1.test();
        System.out.println(" s2 = "+s2);
        s2.test();

        s1.test();
}}
```



```
G:\javaprogs\OOPS\abstraction>java SampleThis
this is sample()
this is sample()
 s1 = SampleThis@3abfe836
this = SampleThis@3abfe836
 s2 = SampleThis@2ff5659e
this = SampleThis@2ff5659e
this = SampleThis@3abfe836
```

- This keyword differentiate between global and local variables.
  For eg :



```
SampleThis s1=new SampleThis();
S.O.P.(s1);
S1.test() ;

SampleThis s2=new SampleThis();
S.O.P.(s2);
S2.test() ;
```

  In the above example firstly this keyword references to s1 object because at that time it is active. But when s2 is active this keyword reference to s2 object.

Constructor Overloading

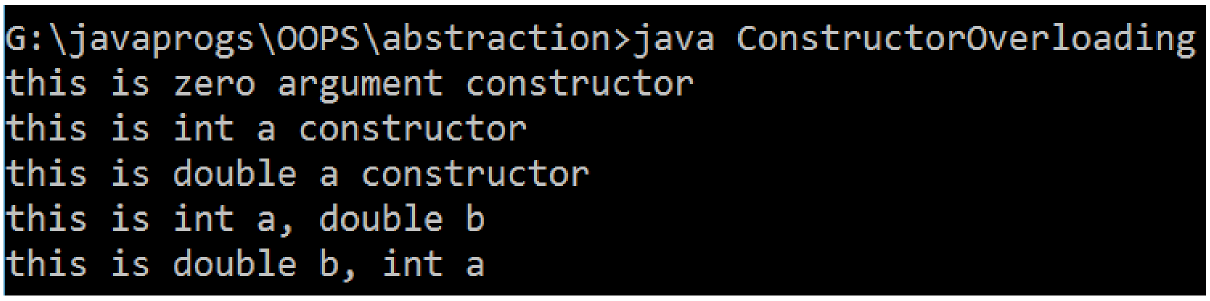
- Developing multiple constructor within the same class which differ in
  (i) No. of arguments
  (ii) Datatypes of arguments
  (iii) Sequence of arguments.

Is called as constructor overloading.

Program

```
class ConstructorOverloading
{
	public ConstructorOverloading()
	{
		System.out.println("this is zero argument constructor");
	}
	public ConstructorOverloading(int a)
	{
		System.out.println("this is int a constructor");
	}
	public ConstructorOverloading(double a)
	{
		System.out.println("this is double a constructor");
	}
```

```
        public ConstructorOverloading(int a, double b)
        {
                System.out.println("this is int a, double b");
        }
        public ConstructorOverloading(double b, int a)
        {
                System.out.println("this is double b, int a ");
        }
        public static void main(String[] args)
        {
                ConstructorOverloading c1=new ConstructorOverloading();
                ConstructorOverloading c2=new ConstructorOverloading(5);
                ConstructorOverloading c3=new ConstructorOverloading(10.2);
                ConstructorOverloading c4=new ConstructorOverloading(5,10.2);
                ConstructorOverloading c5=new ConstructorOverloading(2.5, 7);
        }
}
```



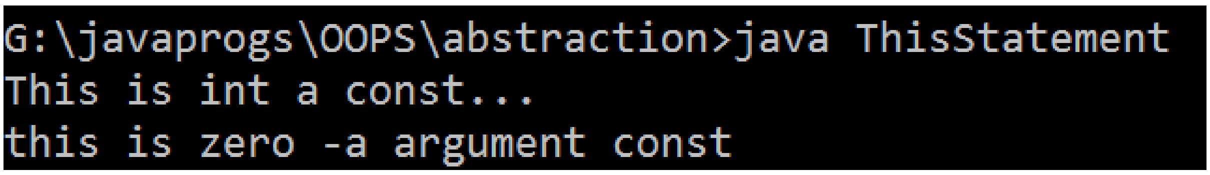
```
G:\javaprogs\OOPS\abstraction>java ConstructorOverloading
this is zero argument constructor
this is int a constructor
this is double a constructor
this is int a, double b
this is double b, int a
```

- Constructor overloading is helpful in providing flexibility for the users to create the objects.
- Constructor overloading is the best example for compile time polymorphism.

This() statement

- It is used to call one constructor from another constructor which are present in same class.

```
        class Sample
        {
                public Sample()
                {
                        this(10); // call the argumented constructor
                        System.out.println("this is zero -a argument const");
                }
                public Sample(int a)
                {
                        System.out.println("This is int a const...");
                }
        }
        class ThisStatement
        {
                public static void main(String[] args) {
                        Sample s=new Sample();
                }
        }
```



```
G:\javaprogs\OOPS\abstraction>java ThisStatement
This is int a const...
this is zero -a argument const
```

- This() statement should be written as first statement within the constructor body.

```
public Sample()
        {
                this(10); // call the argumented constructor
                System.out.println("this is zero -a argument const");
                This(10); if we write like this then it throw error.
        }
        public Sample(int a)
        {
                System.out.println("This is int a const...");
        }
```

- We can not write multiple this() statement within the constructor body.

```
public Sample()
        {
                this(10); // call the argumented constructor
                this(20); // if we write like this it throws an error
                System.out.println("this is zero -a argument const");
        }
        public Sample(int a)
```

```
		{
			System.out.println("This is int a const...");
		}
```

- The called constructor can not call back calling constructor using this() statement because it leads to recursive constructor invocation error.

```
	public Sample()
		{
			this(10); // call the argumented constructor
			System.out.println("this is zero -a argument const");
		}
		public Sample(int a)
		{
			This(); // if we call like this then it throw error.
			System.out.println("This is int a const...");
		}
```

- The constructor can not call itself with this statement, because it leads to recursive constructor invocation.

```
	public Sample()
		{
			this(); // calling itself which throws an recursive
	constructor invocation
			System.out.println("this is zero -a argument const");
		}
		public Sample(int a)
		{
			System.out.println("This is int a const...");
		}
```

Simple this() statement program

```
	class Sample
	{
		public Sample()
		{
			this(10);
			System.out.println("this is zero argument const");
		}
		public Sample(int a)
```