3. Method Overloading in Java

If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.

If we have to perform only one operation, having same name of the methods increases the readability of the program.

Advantage of method overloading

Method overloading *increases the readability of the program*.

Different ways to overload the method

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

> **In java, Method Overloading is not possible by changing the return type of the method only.**

---

1) Method Overloading: changing no. of arguments

```
1. class DisplayOverloading{
2. public void disp(char c)
3. {
4. System.out.println(c);
5. }
6. public void disp(char c, int num)
7. {
8. System.out.println(c+" "+num);
9. }
10. }
11. class Launch{
12. public static void main(String[] args){
13. DisplayOverloading obj = new DisplayOverloading();
14. obj.disp('a');
15. obj.disp('a',10);
16. }
17. }
```

Output:
Kalibermind Academy

```
a
a 10
```

---

2) Method Overloading: changing data type of arguments

```
class DisplayOverloading{
public void disp(char c)
{
System.out.println(c);
}
public void disp( int c)
{
System.out.println(c);
}
}
class Launch{
public static void main(String[] args){
DisplayOverloading obj = new DisplayOverloading();
obj.disp('a');
obj.disp(10);
}
}
```

Output:

```
a
10
```

Q) Why Method Overloading is not possible by changing the return type of method only?

In java, method overloading is not possible by changing the return type of the method only because of ambiguity. Let's see how ambiguity may occur:

```
class Adder{
static int add(int a,int b){return a+b;}
static double add(int a,int b){return a+b;}
}
class TestOverloading3{
public static void main(String[] args){
System.out.println(Adder.add(11,11));//ambiguity
}}
```

Output:

Compile Time Error: method add(int,int) is already defined in class Adder

- System.out.println(Adder.add(11,11)); //Here, how can java determine which sum() method should be called?

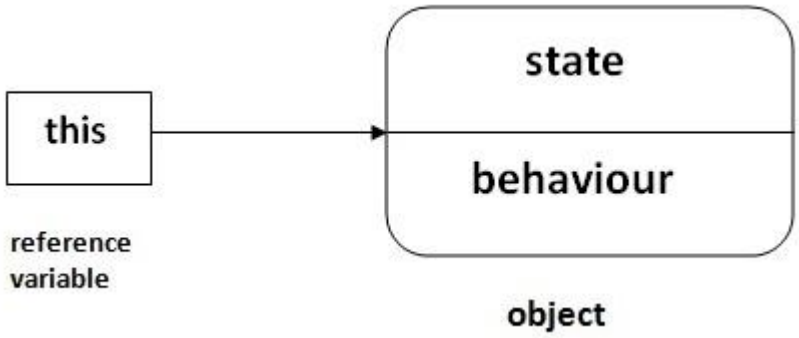
4.this keyword in java

There can be a lot of usage of **java this keyword**. In java, this is a **reference variable** that refers to the current object.

Usage of java this keyword

Here is given the 3 usage of java this keyword.

1. this can be used to refer current class instance variable.
2. this can be used to invoke current class method (implicitly)
3. this() can be used to invoke current class constructor.



1) this: to refer current class instance variable

The this keyword can be used to refer current class instance variable. If there is ambiguity between the instance variables and parameters, this keyword resolves the problem of ambiguity.

➢ Understanding the problem without this keyword

Let's understand the problem if we don't use this keyword by the example given below:

```
1. class Student{
2. int rollno;
3. String name;
4. float fee;
5. Student(int rollno,String name,float fee){
6. rollno=rollno;
```

```
7. name=name;
8. fee=fee;
9. }
10. void display(){System.out.println(rollno+" "+name+" "+fee);}
11. }
12. class TestThis1{
13. public static void main(String args[]){
14. Student s1=new Student(111,"ankit",5000f);
15. Student s2=new Student(112,"sumit",6000f);
16. s1.display();
17. s2.display();
18. }}
```

Output:

```
0 null 0.0
0 null 0.0
```

In the above example, parameters (formal arguments) and instance variables are same. So, we are using this keyword to distinguish local variable and instance variable.

➢ Solution of the above problem by this keyword

```
1. class Student{
2. int rollno;
3. String name;
4. float fee;
5. Student(int rollno,String name,float fee){
6. this.rollno=rollno;
7. this.name=name;
8. this.fee=fee;
9. }
10. void display(){System.out.println(rollno+" "+name+" "+fee);}
11. }
12.
13. class TestThis2{
14. public static void main(String args[]){
15. Student s1=new Student(111,"ankit",5000f);
16. Student s2=new Student(112,"sumit",6000f);
17. s1.display();
18. s2.display();
19. }}
```

Output:

```
111 ankit 5000
112 sumit 6000
```

If local variables(formal arguments) and instance variables are different, there is no need to use this keyword like in the following program:

- **Program where this keyword is not required**

```
1. class Student{
2. int rollno;
3. String name;
4. float fee;
5. Student(int r,String n,float f){
6. rollno=r;
7. name=n;
8. fee=f;
9. }
10. void display(){System.out.println(rollno+" "+name+" "+fee);}
11. }
12.
13. class TestThis3{
14. public static void main(String args[]){
15. Student s1=new Student(111,"ankit",5000f);
16. Student s2=new Student(112,"sumit",6000f);
17. s1.display();
18. s2.display();
19. }}
```

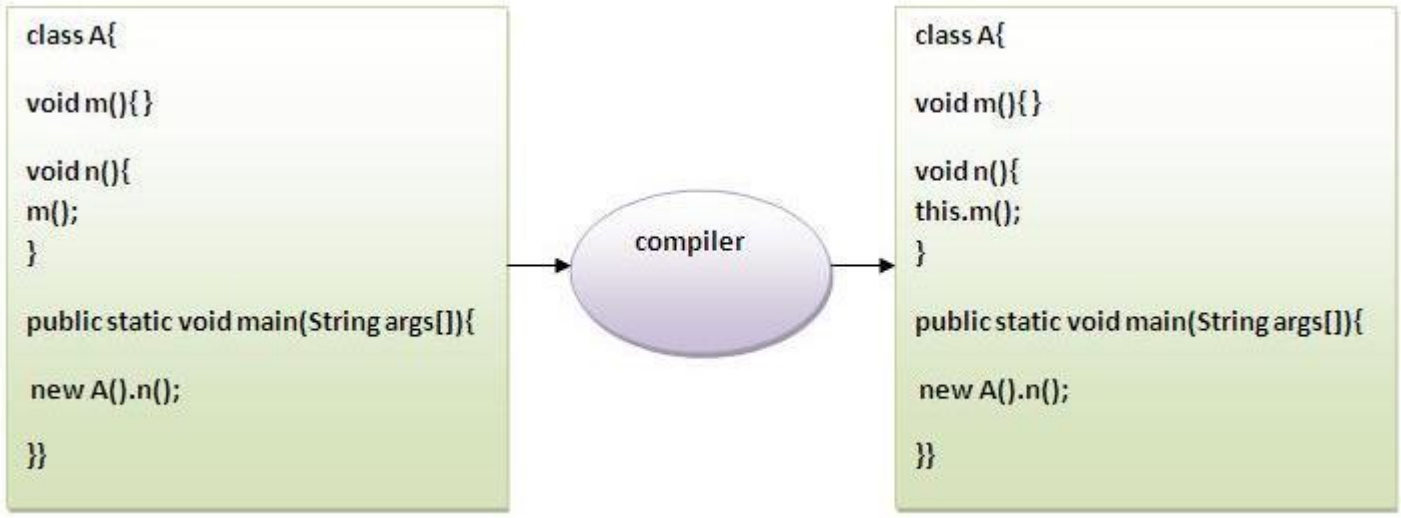
Output:

```
111 ankit 5000
112 sumit 6000
```

It is better approach to use meaningful names for variables. So we use same name for instance variables and parameters in real time, and always use this keyword.

2) this: to invoke current class method

You may invoke the method of the current class by using the this keyword. If you don't use the this keyword, compiler automatically adds this keyword while invoking the method. Let's see the example




```
1. class A{
2. void m(){System.out.println("hello m");}
3. void n(){
4. System.out.println("hello n");
5. //m();//same as this.m()
6. this.m();
7. }
8. }
9. class TestThis4{
10. public static void main(String args[]){
11. A a=new A();
12. a.n();
13. }}
```

Output:

```
hello n
hello m
```

3) this() : to invoke current class constructor

The this() constructor call can be used to invoke the current class constructor. It is used to reuse the constructor. In other words, it is used for constructor chaining.

- **Calling default constructor from parameterized constructor:**

```
1. class A{
2. A(){System.out.println("hello a");}
3. A(int x){
4. this();
```

```
5. System.out.println(x);
6. }
7. }
8. class TestThis5{
9. public static void main(String args[]){
10. A a=new A(10);
11. }}
```

Output:

```
hello a
10
```

➢ Calling parameterized constructor from default constructor:

```
1. class A{
2. A(){
3. this(5);
4. System.out.println("hello a");
5. }
6. A(int x){
7. System.out.println(x);
8. }
9. }
10. class TestThis6{
11. public static void main(String args[]){
12. A a=new A();
13. }}
```

Output:

```
5
hello a
```

➢ Real usage of this() constructor call

The this() constructor call should be used to reuse the constructor from the constructor. It maintains the chain between the constructors i.e. it is used for constructor chaining. Let's see the example given below that displays the actual use of this keyword.

```
1. class Student{
2. int rollno;
3. String name,course;
4. float fee;
```

```
5. Student(int rollno,String name,String course){
6. this.rollno=rollno;
7. this.name=name;
8. this.course=course;
9. }
10. Student(int rollno,String name,String course,float fee){
11. this(rollno,name,course);//reusing constructor
12. this.fee=fee;
13. }
14. void display(){System.out.println(rollno+" "+name+" "+course+" "+fee);}
15. }
16. class TestThis7{
17. public static void main(String args[]){
18. Student s1=new Student(111,"ankit","java");
19. Student s2=new Student(112,"sumit","java",6000f);
20. s1.display();
21. s2.display();
22. }}
```

Output:

```
111 ankit java null
112 sumit java 6000
```

Rule: Call to this() must be the first statement in constructor.

```
1. class Student{
2. int rollno;
3. String name,course;
4. float fee;
5. Student(int rollno,String name,String course){
6. this.rollno=rollno;
7. this.name=name;
8. this.course=course;
9. }
10. Student(int rollno,String name,String course,float fee){
11. this.fee=fee;
12. this(rollno,name,course);//C.T.Error
13. }
14. void display(){System.out.println(rollno+" "+name+" "+course+" "+fee);}
15. }
16. class TestThis8{
```

```
17. public static void main(String args[]){
18. Student s1=new Student(111,"ankit","java");
19. Student s2=new Student(112,"sumit","java",6000f);
20. s1.display();
21. s2.display();
22. }}
```

Output:

Compile Time Error: Call to this must be first statement in constructor