Implicit Passing of Object:

When a method of a class is called, a reference of the associated object is passed implicitly into the method. This object can be found using ***this** pointer inside the method. Any change in the object in the method will impact the calling part.

- For unary operation like **++a** or **a++**, there is only one object associated with the operator and that object is passed into the method.
- For binary operation like **a+b** or **a += b**, the object left to the operator is passed into the method.

Example Program:

```
#include <iostream>
using namespace std;

class Coord{
     int x, y;
public:
    Coord(int a=0, int b=0){ x = a; y = b;}

    Coord operator ++(){
        x++;
        y++;
    }

    void operator += (Coord const &ob){
        x += ob.x;
        y += ob.y;
    }

    void show() const{
        cout << "(" << x << "," << y << ")" << endl;
    }
};
```

```
int main(){
    Coord a(10, 20), b(2, 3);

    a += b;
    a.show();
    ++a;
    a.show();
    b = ++a;
    b.show();

    return 0;
}
```

Output:

```
(12,23)
(13,24)
(-1216260480.32759)
```

Discussion on Result: In statement, **a += b**; the reference object **a** will be passed to the method implicitly. The changes inside the implicitly passed object will be reflected in the main function. No need to return the implicit object. Similarly, for the statement **++a**; the changes in the implicitly passed object will be reflected in the main function.

However, in the statement, **b = ++a**; object **a** will be changed; but as the method does not return any value, nothing but the garbage value will be returned and assigned to object **b**.

If we write **void operator ++ (){ x++; y++; }** instead of **Coord operator ++ (){ x++; y++; }** in the **Coord** class, then statement **b = ++a**; in the main function will generate error.

Prefix Increment Operator:

For only **++a**; operation, the following method is sufficient.

```
void operator ++(){
    x++;
    y++;
}
```

However, for the operation of **b = ++a**; this method will not work. The method must return ***this** object. Hence, the correct method will be as follows:

```
Coord operator ++(){
    x++;
    y++;
    return *this;
}
```

The correct method increments the values and the incremented values are then returned to object **b** in the following statement, **b = ++a**;

Friend function of **++a** operation is as follows:

```
Coord operator ++(Coord &c){
    c.x++;
    c.y++;
    return c;
}
```

Postfix Increment Operator:

For only **a++**; operation, the following method is sufficient.

```
void operator ++(int unused){
    x++;
    y++;
}
```

However, for the operation of **b = a++**; this method will not work. The method at first need to store the previous value of ***this** implicit object into a temporary object and then the implicit ***this** object will be incremented. Increment of ***this** object will change the values of **a**. After that the temporary object will be returned. Hence, object **b** will get previous values of object **a**; not the current or changed values. The correct program is given below:

```
Coord operator ++(int unused){
    Coord temp = *this;
    x++;
    y++;
    return temp;
}
```

Friend function of **a++** operation is as follows:

```
Coord operator ++(Coord &c, int unused){
    Coord temp = c;
    c.x++;
    c.y++;
    return temp;
}
```

Complete methods inside the body of an Interface:

An interface in Java may contains three types of full methods:

1. default method;
2. private method; and
3. static method.

Sample Program:

```
interface InterfaceDemo{
    public double PI = 3.14159;
    public double volume();
    default double circleArea(double radius) {
        System.out.println(getName());
        return PI*radius*radius;
    }
    private String getName(){
        return "Area is calculated using default method.";
    }

    public static void display(String str, double val){
        System.out.println(str+" volume : "+val);
    }
}

class Cone implements InterfaceDemo{
    private String name;
    private double height;
    private double radius;
    Cone(double nh, double nr){
        height = nh;
        radius = nr;
    }
    public double volume(){
        return height * circleArea(radius) / 3;
    }
}

class Cylindar implements InterfaceDemo{
    private String name;
```

```
    private double height;
    private double radius;
    Cylindar(double nh, double nr){
        height = nh;
        radius = nr;
    }
    public double volume(){
        return height * circleArea(radius);
    }
}

public class Main {
    public static void main(String[] args) {
        Cylindar cylindar = new Cylindar(5.6, 4.2);
        Cone cone = new Cone(5.6, InterfaceDemo.PI);

        InterfaceDemo.display("Cylindar", cylindar.volume());
        InterfaceDemo.display("Cone", cone.volume());
    }
}
```

==========