Does overloading work with Inheritance?

Difficulty Level : Medium • Last Updated : 21 Sep, 2021

If we have a function in base class and a function with same name in derived class, can the base class function be called from derived class object? This is an interesting question and as an experiment predict the output of the following **C++** program.

C++

```
#include <iostream>
using namespace std;
class Base
{
public:
    int f(int i)
    {
        cout << "f(int): ";
        return i+3;
    }
};
class Derived : public Base
{
public:
    double f(double d)
    {
        cout << "f(double): ";
        return d+3.3;
    }
};
int main()
{
    Derived* dp = new Derived;
    cout << dp->f(3) << '\n';
    cout << dp->f(3.3) << '\n';
    delete dp;
    return 0;
}
```

The output of this program is:

```
f(double): 6.3
f(double): 6.6
```

Instead of the supposed output:

```
f(int): 6
f(double): 6.6
```

Overloading doesn't work for derived class in C++ programming language. There is no overload resolution between Base and Derived. The compiler looks into the scope of Derived, finds the single function "double f(double)" and calls it. It never disturbs with the (enclosing) scope of Base. In C++, there is no overloading across scopes – derived class scopes are not an exception to this general rule. (See this for more examples)

Reference: technical FAQs on www.stroustrup.com

Now consider **Java** version of this program:

Java

```
class Base
{
    public int f(int i)
    {
        System.out.print("f (int): ");
        return i+3;
    }
}
class Derived extends Base

    public double f(double i)
    {
        System.out.print("f (double) : ");
        return i + 3.3;
    }
}
class myprogram3
{
    public static void main(String args[])
    {
        Derived obj = new Derived();
        System.out.println(obj.f(3));
        System.out.println(obj.f(3.3));
    }
}
```

The output of the above program is:

```
f (int): 6
f (double): 6.6
```

So in Java overloading works across scopes contrary to C++. Java compiler determines correct version of the overloaded method to be executed at compile time based upon the type of argument used to call the method and parameters of the overloaded methods of both these classes receive the values of arguments used in call and executes the overloaded method.

Finally, let us try the output of following **C#** program:

C#

```
using System;
class Base
{
    public int f(int i)
    {
        Console.Write("f (int): ");
        return i + 3;
    }
}
class Derived : Base
{
    public double f(double i)
    {
        Console.Write("f (double) : ");
        return i+3.3;
    }
}
class MyProgram
{
    static void Main(string[] args)
    {
        Derived obj = new Derived();
        Console.WriteLine(obj.f(3));
        Console.WriteLine(obj.f(3.3));
        Console.ReadKey(); // write this line if you use visual stud
    }
}
```

Note: Console.ReadKey() is used to halt the console. It is similar to getch as in C/C++.
The output of the above program is:

```
f(double) : 6.3
f(double):  6.6
```

Instead of the assumed output

```
f(int) : 6
f(double) : 6.6
```

Explanation: Here, we are creating the object of the derived class so the compiler will give preference to the derived class first and will perform implicit type casting if needed. So as soon as the compiler comes at *Console.WriteLine(obj.f(3));* this line of code it will check for the parameters compatibility. Here the 3 is *int* which is compatible with the *double* of *derived class function f*. So compiler will perform the implicit type conversion of *int to double*. Hence the output *f(double) : 6.3* will come.

Now when the compiler goes at *Console.WriteLine(obj.f(3.3));*, again it will give the preference to the derived class and it finds callable. So it will evaluate the derived class function *f*. Hence the output *f(double): 6.6* will come.

ow Let's take another case where we are putting the base class function f into erived class and vice-versa as shown below:

C#

```
using System;
class Base
{
    public double f(double i)
    {
        Console.Write("f (double) : ");
        return i+3.3;
    }
}

class Derived : Base
{

    public int f(int i)
    {
        Console.Write("f (int): ");
        return i + 3;
    }

}

class MyProgram
{
    static void Main(string[] args)
    {
        Derived obj = new Derived();
        Console.WriteLine(obj.f(3));
        Console.WriteLine(obj.f(3.3));
        Console.ReadKey(); // write this line if you use visual stud
    }
}
```

Output:

```
f (int): 6
f (double) : 6.6
```

Are you shocked to see the expected output? How is it possible?

ell, we have an answer to these questions. As we have created the object of erived class so C# compiler will give first preference to derived and if it does not find any compatibility then it goes for the base class. So when the compiler comes at *Console.WriteLine(obj.f(3));* line of code, it will check the *derived class method f* and it finds callable hence execute this and output comes *f (int): 6*. Now when the *Console.WriteLine(obj.f(3.3));* line of code will execute it will find that the derived class method is not suitable as 3.3(double) is not compatible with the int data type hence compiler will prefer now to the base class and there it found the best match and it execute that one. So the output for that one will be *f (double) : 6.6*.

The reason is the same as explained in the C++ program. Like C++, there is no overload resolution between class Base and class Derived. In C#, there is no overloading across scopes derived class scopes are not an exception to this general rule. This is same as C++ because C# is designed to be much closer to C++, according to the Anders hejlsberg the creator of C# language.

This article is contributed by **Pravasi Meet**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Next

Can main() be overloaded in C++?