Why doesn't overloading work for derived classes?

eenshot stion (in many variations) are usually prompted by an example like this:

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

class B {
public:
        int f(int i) { cout << "f(int): "; return i+1; }
        // ...
};

class D : public B {
public:
        double f(double d) { cout << "f(double): "; return d+1.3; }
        // ...
};

int main()
{
        D* pd = new D;

        cout << pd->f(2) << '\n';
        cout << pd->f(2.3) << '\n';
}
```

which will produce:

```
f(double): 3.3
f(double): 3.6
```

rather than the

```
f(int): 3
f(double): 3.6
```

that some people (wrongly) guessed.

In other words, there is no overload resolution between D and B. The compiler looks into the scope of D, finds the single function "double f(double)" and calls it. It never bothers with the (enclosing) scope of B. In C++, there is no overloading across scopes - derived class scopes are not an exception to this general rule. (See D&E or TC++PL3 for details).

But what if I want to create an overload set of all my f() functions from my base and derived class? That's easily done using a using-declaration:

```
class D : public B {
public:
        using B::f;     // make every f from B available
        double f(double d) { cout << "f(double): "; return d+1.3; }
        // ...
};
```

Give that modification, the output will be