Inheritance: More on Constructors

What if I don't want to call the default constructor all the time? What if I want to use both the base and derived constructors?

That's where some new notation enters in.

Consider these renovations to our classes:

```
// class spec for brian class
class brian {
public:
        brian(string x) { cout << "Brian says, "<< x << endl; }
};
```

```
// class spec for jeff class
class jeff : public brian {
public:
        jeff(string x, string y) : brian(x) { cout << "Jeff says, " << y << endl; }
};
```

```
// class spec for steve class
class steve : public jeff {
public:
        steve(string x, string y, string z) : jeff(x, y) {
                cout << "Steve says, " << z << endl; }
};
```

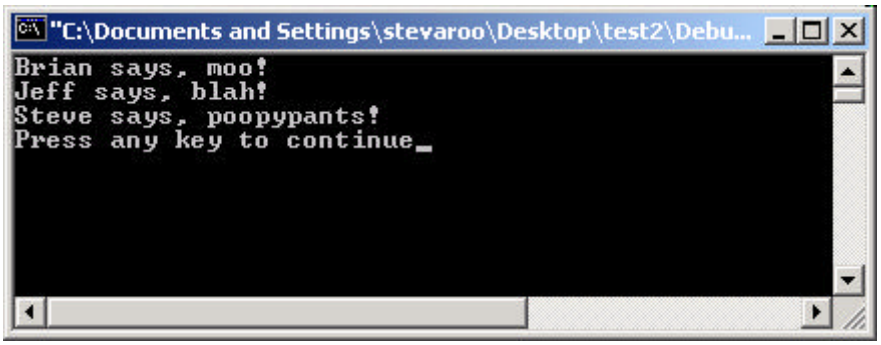
Now, what does the following output? Why?

```
//main file
#include "brian.h"
#include "jeff.h"
#include "steve.h"

int main ( ) {
        steve test("moo!", "blah!", "poopypants!");
        return 0;
}
```

Inheritance: More on Constructors, part 2

Well, it outputs the following:



General rule of thumb:

If there is no alternate constructor from a base class specified in your IMPLEMENTATION of a derived class constructor, then the default will always run for the base class every time you instantiate an instance of the derived class. Remember that the derived class 'is a' instance of the base class!

Example: the classes I showed earlier!

To SPECIFY an alternate constructor to run, you must use the following syntax:

Constructor name, etc

Constructor arguments

The call to the base class constructor, passing in arguments from the call to the derived constructor.

```
steve :: steve(string x, string y, string z) : jeff(x, y) { . . . . }
```

Note that in order to use this, you must have a corresponding constructor implemented in the base class. Otherwise, the compiler won't know what to call.

This gives you precise control over what constructors are called and with what arguments. . .nifty!