Base-class Constructors

Now that you've learned about inheritance and inherited members, let's look at how **derived-class constructors** are written. We'll with our simple "finger-exercise" example that lets you concentrate on one piece of the inheritance puzzle at a time. You can re-open it directly from **Replit**, or you can **click the Running Man** to open my copy, and **Fork** it again.

A constructor **must** have the same name as the class, so, when you create a new class, it **cannot inherit any of the base class constructors**. Instead, it defines new ones.

To see how that works, modify the `Person` class, which now uses only the **synthesized default constructor** that is automatically written by your compiler, when you don't supply any explicit constructors. In `person.h` add this code:

```
class Person
{
public:
    Person();
    Person(const std::string& pname);
   . . .
};
```

In `person.cpp` add an implementation that prints a message so you can keep track of which constructor is called. The working constructor should use its `string` parameter to initialize the `m_name` data member in addition to **printing a diagnostic message**.

```
Person::Person() { cout << "Calling Person()" << endl; }

Person::Person(const string& pname)
{
    cout << "Calling Person(" << pname << ")" << endl;
    name = pname;
}
```

This course content is offered under a CC Attribution Non-Commercial license. Content in this course can be considered under this license unless otherwise noted.