Virtual Member Functions

Now that you've learned about inheritance and constructors, let's take a look at how derived-class member functions may be **redefined or overridden**. Open the **Repl** from Lesson **15A** and we'll continue with our simple "finger-exercise" example that lets you concentrate on one piece of the inheritance puzzle at a time.

A derived class may override member functions in the base class. The base class must permit that by using the keyword `virtual` on the prototype. Let's see how that works by modifying the `Person` class to add a new `virtual toString()` member function and a `virtual` destructor like this:

```
class Person
{
public:
    . . .
    virtual std::string toString() const;
    virtual ~Person() = default;
private:
    std::string name;
};
```

It is up to the **base class designer** to decide which member functions **may be** overridden and which may not. Member functions which allow derived classes to override them **should be preceded** with the keyword `virtual`.

As soon as you add a single `virtual` function, you should add a **`virtual` destructor** as shown in the `Person` class header. This uses the `=default` keyword to keep the synthesized destructor written by the compiler.