CEE432/CEE532/MAE541
Developing Software for Engineering Applications

Lecture 14: More on Objects 303
(Chapter 13)

Polymorphism

- What happens if
  - the client code does not know ahead of time what specific cross-sectionals shape are going to be used during a program run
  - new cross-sectional shapes are to be added
  - undefined situations need to be detected.

Polymorphism

- Polymorphism is the ability of different objects to respond in their own way to the same message by means of virtual functions or late binding or dynamic binding.

Virtual Functions

- In the base class (class definition) declare the required function as a virtual function.

```
virtual void DisplayDimensions ();

virtual void ComputeProperties ();
```

- We could define these functions in the base class.

```
void CXSType::DisplayDimensions ()
{
  std::cout << "Invalid call to base class DisplayDimensions(). "
            << "Do not know what derived class dimensions mean.\n";
}
```

Virtual Functions

- In the derived class declare the required function as a virtual function.

```
virtual void DisplayDimensions ();

virtual void ComputeProperties ();
```

- Now define the function without the virtual keyword.

```
void CISection::DisplayDimensions ()
{
……….
}
```

Virtual Functions

- If a function is declared to be virtual and the function is redefined in the derived class, then for any object associated with the derived class, the derived class version of the virtual function is used not the base class version.
- Declare the function as `public` function so that it can be called directly by the client code.

Example 13.3.1
Using Inheritance with Virtual Member Functions

Pure Virtual Function

- These are declared in the base class when the base class cannot meaningfully implement the functions. Such a base class is called an *abstract base class*.

```
virtual return_type functionname (….) = 0;
```

- The derived class must define the functions for the program to work.

Pure Virtual Functions

- It is perfectly valid to assign a derived class object to a base class object. The information that is available only in the derived class cannot be stored and made available in the base class (slicing problem).
- Solution: Manipulate derived class information via a base class object.

Example 13.3.2
Using Inheritance with Pure Virtual Member Functions

Pure Virtual Functions

- Derived class destructors will not be called unless we use virtual destructors.
- In other words, if a base class destructor is tagged virtual, then the derived class destructor is called first followed by the base class destructor.
- If the destructor for the base class is tagged virtual, then automatically destructors for the derived classes are tagged virtual.

Example 13.3.3
Using Inheritance with Pure Virtual Member Functions and Virtual Destructors

Run-Time Typing Information (RTTI)

- What can we do if at run-time we need to know the type of the object so that an appropriate action can be taken?
- C++ provides a casting operator known as `dynamic_cast` that can be used to extract the information at run-time.
- One can also use the `typeid` operator.

Functors

- Functors are *functions with a state*.
- When designing general purpose or library components, it is often necessary or desirable to put in hooks for calling unknown objects.

Summary

- Inheritance provides the mechanism to develop new functionalities based on some generic requirements.
- Dynamic binding lets the compiler decide at run time what specific function to call based on the object's dynamic type.
- Inherited objects are composed of base class parts and derived class parts. They are constructed, copied and assigned by manipulating the base part of the object before the derived part.