Object-Oriented Programming

Lesson 11

Abstract classes
Pure virtual functions

Abstract classes

Often in a design, you want the base class to present only an interface for its derived classes. That is, you don’t want anyone to actually create an object of the base class, only to upcast to it so that its interface can be used.

This is accomplished by making that class abstract, which happens if you give it at least one pure virtual function.

You can recognize a pure virtual function because it uses the virtual keyword and is followed by = 0. If anyone tries to make an object of an abstract class, the compiler prevents them.

```
virtual double Area () const = 0;
```

If a class only contains pure virtual functions, it is called pure abstract class.

Abstract classes and inheritance

When an abstract class is inherited, all pure virtual functions must be implemented, or the inherited class becomes abstract as well.

Creating a pure virtual function allows you to put a member function in an interface without being forced to provide a possibly meaningless body of code for that member function. At the same time, a pure virtual function forces inherited classes to provide a definition for it.

The compiler will reserve a slot for a pure virtual function in the VTABLE, but will not put an address in that particular slot. Even if only one function in a class is declared as pure virtual, the VTABLE is incomplete.

If the VTABLE for a class is incomplete, the compiler will not be able to create instances of that class

=> service for the user:

you ensure that the client programmer cannot misuse an abstract class.

Pure virtual functions

Usually, pure virtual functions do not have definitions in the base class.

It’s however possible to provide a definition for a pure virtual function in the base class.

You’re still telling the compiler not to allow objects of that abstract base class, and the pure virtual functions must still be defined in derived classes in order to create objects. However, there may be a common piece of code that you want some or all of the derived class definitions to call rather than duplicating that code in every function.

ieeta

Pure virtual destructors

Pure virtual destructors are legal in Standard C++.

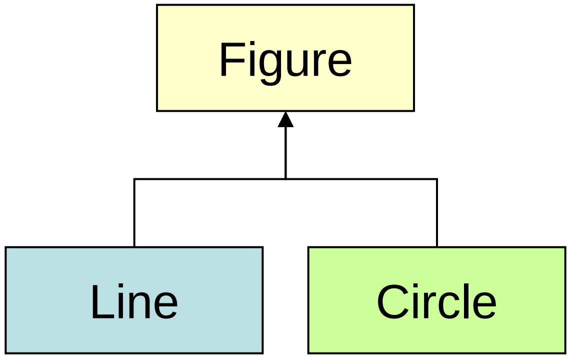
There are added constraints when using them:

1) you must provide a function body for the pure virtual destructor

2) when you inherit a class from one that contains a pure virtual destructor, you are ***not*** required to provide a definition of a pure virtual destructor in the derived class.

Pure virtual functions

```
class Figure  //abstract class
{
   //...................
public:
   virtual void draw() = 0;
   virtual void move(int, int) = 0;
   virtual void copy(int, int) = 0;
};
```



```
class Line: public Figure
{
   int x1, y1, x2, y2;
public:
   void draw();
   void move(int, int);
   void copy(int, int);
   // .................
};
```

```
class Circle : public Figure
{  int diameter;
   int cx, cy;
public:
   void draw();
   void move(int, int);
   void copy(int, int);
   // .................
};
```

Bibliography

Bruce Eckel, Thinking in C++, 2nd edition, MindView, Inc., 2003

=> Chapter 15