Shape class, but any derived class that inherits from Shape must implement them.

Now let's look at an example of a derived class that inherits from the Shape abstract class and provides its own implementation for the "draw" and "area" functions:

csharp Copy code

```
class Circle : public Shape {
  public:
    Circle(double r) : radius(r) {}
    void draw() {
        // implementation for drawing a circle
    }
    double area() {
        return 3.14 * radius * radius;
    }
  private:
    double radius;
};
```

In this example, the Circle class inherits from the Shape class using the "public" access specifier. It provides its own implementation for the "draw" function, which will draw a circle, and the "area" function, which calculates the area of a circle.

Notice that the Circle class provides implementations for both of the pure virtual functions defined in the Shape class. This is necessary, because any class that inherits from an abstract class must implement all of the pure virtual functions defined in the base class.

By defining an abstract class in this way, you can create a hierarchy of classes that share a common interface, but have different implementations for that interface. This can be useful for organizing and simplifying your code, and for making it more modular and reusable.