New chat

`perimeter()`. Any class that inherits from `Shape` must implement these functions, or else it will also be an abstract class.

Let's say we want to create a `Rectangle` class that inherits from `Shape`. Here's an example implementation:

c++ Copy code

```
class Rectangle : public Shape {
private:
    double length_;
    double width_;
public:
    Rectangle(double length, double width) : length_(length), width_(widt
    double area() const override { return length_ * width_; }
    double perimeter() const override { return 2 * (length_ + width_); }
};
```

In this example, `Rectangle` inherits from `Shape` and provides implementations for both `area()` and `perimeter()`. We can now create instances of `Rectangle` and call the interface functions on them:

c++ Copy code

```
Rectangle r(3, 4);
std::cout << "Area: " << r.area() << std::endl; // prints "Area: 12"
std::cout << "Perimeter: " << r.perimeter() << std::endl; // prints "Peri
```

Using interfaces like this can be useful for creating a consistent interface across different classes. For example, we could create other classes that inherit from `Shape`, like `Circle` or `Triangle`, and use them interchangeably in functions that expect a `Shape` object.