【C++】Day73(2)

Class	C++
Date	@March 14, 2022
Material	
Series Number	
Summary	

【Ch15】OOP

15.4 Abstract Base Classes

Imagine that we want to extend our bookstore classes to support several discount strategies.

We might offer a discount for purchases up to a certain quantity and then charge the full price thereafter. Or we might offer a discount for purchases above a certain limit but not for purchases up to that limit.

Each of these discount strategies is the same in that it requires a quantity and a discount amount. We might support these differing strategies by defining a new class named `Disc_quote` to store the quantity and the discount amount.

Classes, such as `Bulk_item` that represent a specific discount strategy will inherit from `Disc_quote`.

Each of the classes will implement its discount strategy by defining its own version of `net_price`.

We could define `Disc_quote` without its own version of `net_price` because it does not correspond to any particular discount strategy.

Pure Virtual Functions

We'd like to prevent users from creating `Disc_quote` objects at all. This class represents the general concept of a discounted book, not a concrete discount strategy.

We can enforce this design intent-and make it clear that there is no meaning for `net_price`-by defining `net_price` as a pure virtual function.

Unlike ordinary virtuals, a pure virtual function does not have to be defined. We specify that a virtual function is a pure virtual by writing `= 0` in place of a function body. The = 0 may appear only on the declaration of a virtual function in the class body.

```
// class to hold the discount rate and quantity
// derived class will implement pricing strategies using these data
class Disc_quote : public Quote {
public:
  Disc_quote() = default;Disc_quote(const std::string *&book, double price, std::size_t qty, double disc):
    Quote(book, price), quantity(qty), discount(disc) {}
  double net_price (std::size_t) const = 0;
protected:
  std::size_t quantity = 0;
  double discount = 0.0;
};
```

Although we cannot define objects of this type directly, constructors in classes derived from `Disc_quote` will use the `Disc_quote` constructors to construct the `Disc_quote` part of their objects.

Classes with Pure Virtuals are Abstract Base Classes

A class containing (or inheriting without overriding) a pure virtual function is an abstract base class. An abstract base class defines an interface for subsequent classes to override.

We cannot (directly) create objects of a type that is an abstract class.

Because `Disc_quote` defines `net_price` as a pure virtual, we cannot define objects of type `Disc_quote`. We can define objects of classes that inherit from `Disc_quote`, so long as those classes override `net_price`.

```
Disc_quote discounted; // error: cannot define a Disc_quote object
Bulk_quote bulk; // ok: Bulk_quote has no pure virtual functions.
```

Classes that inherit from `Disc_quote` must define `net_pric` `e` or those classes will be abstract as well.

Note: We may not create objects of a type that is an abstract base class.

A Derived Class Constructor Initializes its Direct Base Class Only

Now we can reimplement `Bulk_quote` to inherit from `Disc_quote`:

```
// the discount kicks in when a specified number of copies of the same book are sold
// the discount is expressed as a fraction to use to reduce the normal price
class Bulk_quote : public Disc_quote {
public:
  Bulk_quote() = default;
  Bulk_quote(const std::string &book, double price, std::size_t qty, double disc) :
    Disc_quote(book, price, qty, disc) {}
  // overrides the base version to implement the bulk purchase discount policy
  double net_price(std::size_t) const override;
};
```

Exercise

Exercise 15.15: Define your own versions of `Disc_quote` and `Bulk_quote`.

Exercise 15.16: Rewrite the class representing a limited discount strategy, which you wrote for the exercises in § 15.2.2 (p. 601), to inherit from `Disc_quote`.

See 15_15.cpp for code