【C++】Day60

Class	C++
Date	@February 21, 2022
Material	
Series Number	
Summary	Example: Message and Folder

【Ch13】Copy Control

13.4 A Copy-Control Example

As an example of a class that needs copy control in order to do some bookkeeping, we'll sketch out two classes that might be used in a mail-handling application.

These classes, `Message` and `Folder`, represent, respectively, email message and directories in which a message might appear. Each `Message` can appear in multiple `Folders`. However, there will be only one copy of the contents of any given Message.

To keep track of which `Messages` are in which Folders, each `Message` will store a set of pointers to the `Folders` in which it appears, and each `Folder` will contain a set of pointers to its `Message`s.

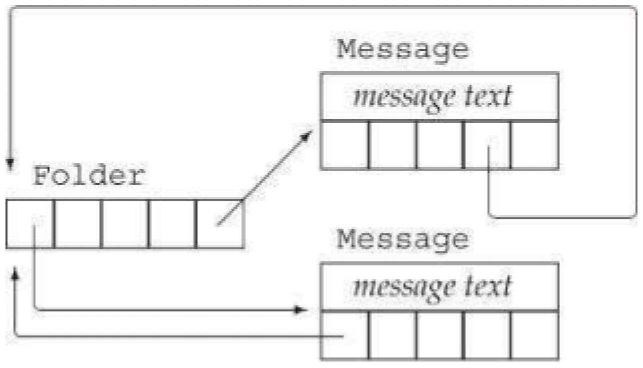


Figure 13.1. Message and Folder Class Design

Our `Message` class will provide `save` and `remove` operations to add or remove a `Message` from a specified `Folder`. To create a new `Message`, we will specify the contents of the message but no Folder. To put a Message in a particular Folder, we must call save.

The Message Class

```
class Message {
  friend class Folder;
public:
  // folders implicitly initializers to the empty set
  explicit Message(const string& str = "") : contents(str) {}
  // copy control to manage pointers to this Message
  Message(const Message&);
  Mesage& operator=(const Message&);
  ~Message();
  // add/remove this Meesage from the specified Folder's set of messages
  void save(Folder&);
  void remove(Folder&);

private:
  string contents; // actual message
  set<Folder*> folders; // Folders that have this Message
  // utility functions used by copy constructor, assignment, and destructor
  // add this Message to the Folders that point to the parameter
  void add_to_Folders(const Message&);
  // remove this Message from every  Folder in folders
  void remove_from_Folders();
};
```

The save and remove Members

```
void Message::save(Folder &f) {
  folders.insert(&f); // add the given Folder to our list of Folders
  f.addMsg(this); // add this Message to f's set of Messages
}

void Message::remove(Folder &f) {
  folders.erase(&f); // take the given Folder out of our list of Folders
  f.remMsag(this); // remove this Message to f's set of Messages
}
```

Copy Control for the Message Class

When we copy a `Message`, the copy should appear in the same `Folders` as the original Message. As a result, we must traverse the set of Folder pointers adding a pointer to the new `Message` to each `Folder` that points to the original Message.

```
// add this Message to Folders that point to m
void Message::add_to_Folders(const Message &m) {
  for(auto f : m.folders)
    f->addMsg(this); // add a pointer to this Message to that folder
}
```

The Message copy constructor copies the data members of the given object:

```
Message::Message(const Message &m) : contents(m.contents), folders(m.folders) {
  add_to_Folders(m);
}
```

The Message Destructor

When a Message is destroyed, we must remove this Message from the Folders that point to it.

```
// remove this Message from the corresponding Folders
void Message::remove_from_Folders() {
  for(auto f : folders)
    f->remMsg(this);
}

Message::~Message() {
  remove_from_Folders
}
```

Message Copy-Assignment Operator

```
Message& Message::operator=(const Message &rhs) {
  remove_from_Folders(); // update existing Folders
  contents = rhs.contents;
  folders = rhs.folders; // copy Folder pointers from rhs
  add_to_Folders(rhs); // add this Message to those Folders in rhs
  return *this;
}
```

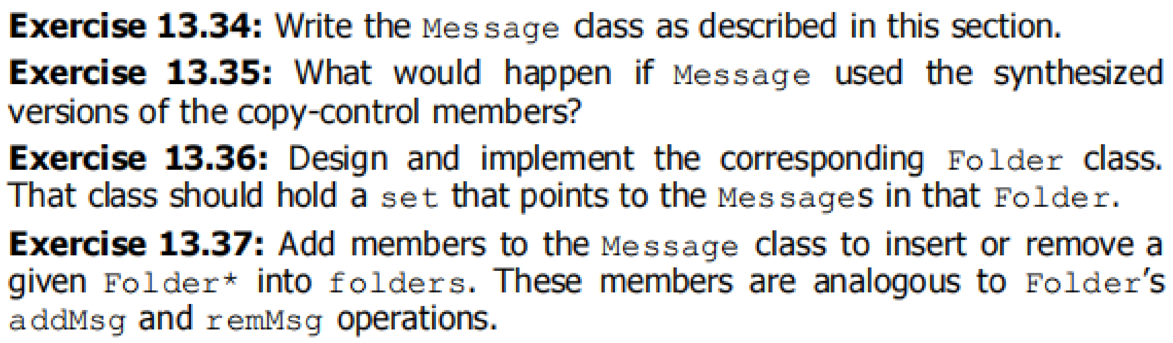
Define swap

```
void swap(Message &m1, Message &m2) {
  for(auto f : m1.folders)
    f->remMsg(&m1);
  for(auto f : m2.folders)
    f->remMsg(&m2);

  swap(m1.contents, m2.contents);
  swap(m1.folders, m2.folders);

  for(auto f : m1.folders)
    f->addMsg(&m1);
  for(auto f : m2.folders)
    f->addMsg(&m2);
}
```

Exercise


Exercise 13.34: Write the `Message` class as described in this section.

Exercise 13.35: What would happen if `Message` used the synthesized versions of the copy-control members?

Exercise 13.36: Design and implement the corresponding `Folder` class. That class should hold a `set` that points to the `Message`s in that `Folder`.

Exercise 13.37: Add members to the `Message` class to insert or remove a given `Folder*` into `folders`. These members are analogous to `Folder`'s `addMsg` and `remMsg` operations.

See 13_34 for code