《面向对象的程序设计实验课》期末考试

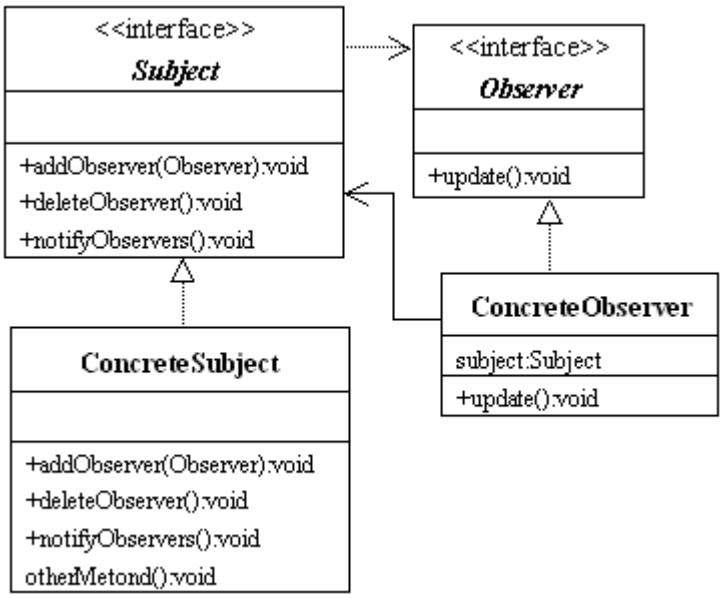
利用“观察者模式”实现求职中心应用程序

1．观察者模式（Observer Pattern）又被称为发布-订阅模式（Publish/Subscribe）。这种模式定义了不同对象间一对多的依赖关系，当对象 A 的状态发生改变时主动发出通知，所有依赖于 A 的其他对象都会自动得到通知，并做出响应。此种模式通常被用来实现事件处理系统。

2．在许多设计中，经常涉及到多个对象都对一个特殊对象 A 中的数据变化感兴趣，而且这些对象都希望跟踪特殊对象 A 中的数据变化。

例如，求职者对“求职中心”的职位需求信息非常关心，很想跟踪“求职中心”中职位需求信息的变化。而现实生活中的实现方式是，“求职中心”让求职者把个人信息登记下来，当出现新的职位需求时，“求职中心”会及时通知全体求职者，而求职者会根据不同的职位需求做出不同的响应：应聘或忽略

3. 观察者模式的结构中包含 4 种角色：主题 Subject、具体主题 ConcreteSubject、观察者 Observer、具体观察者 ConcreteObserver，模式的 UML 类图表示如下。



4．编写程序，模拟求职者在“求职中心”进行登记，“求职中心”发布职位需求信息，求职者决定是否应聘的过程。

（1）创建“求职中心”对象，其成员变量可以保存所有的求职者；

（2）创建不同类型的求职者对象（**应届大学毕业生 / 有工作经验的求职者**），并在“求职中心”中进行注册登记；

（3）“求职中心”发布新的职位需求信息，并及时通知所有注册过的求职者。

（4）求职者能够自动接收信息，并做出响应。

7. 题目要求

（1）创建新的工程项目 Findjob；

（2）为每个类编写单独的头文件和源文件，例如 JobCenter.h 和 JobCenter.cpp；

（3）主程序对应的源文件名为 Findjob.cpp，里面包含 main()函数。

8. 编写程序，测试观察者模式的实现，可以自由扩展程序。

```
class Subject {
public:
    virtual void addObserver(Observer *pObserver) = 0;     // 添加观察者对象
    virtual void notifyAll() = 0;    // 通知所有的观察者，让他们接收信息
};
class JobCenter: public Subject {
public:
    void addObserver(Observer *pObserver);      // 添加求职者对象
    void publishMessage(const string &message); // 发布职位需求信息
    void notifyAll();     // 通知所有的求职者，让他们接收职位需求信息
private:
    string message;       // 职位需求信息
    bool statusChanged;   // 对象状态是否发生变化，即是否出现新的职位需求信息
    int numObserver;      // 当前求职者的数量，也是下面指针数组的最有一个元素的下标
    Observer *list[100];  // 指针数组，保存Observer对象的指针
};
class Observer {
public:
    Observer(const string &name);
    virtual void answerPhone(const string &message) = 0; // 接收职位需求信息
protected:
    string name;          // 观察者的名字
};
class Student: public Observer { // 应届大学毕业生
public:
    Student(const string &name, Subject *subject); // 在求职中心登记
    void answerPhone(const string &message);
};
class Experienced: public Observer { // 有工作经验的求职者
public:
    Experienced(const string &name, Subject *subject);
    void answerPhone(const string &message);
};
#include <string>
int main() {
    JobCenter *jobCenter = new JobCenter();    // 创建“求职中心”

    // 创建求职者，并在求职中心登记
    Student *zhang = new Student("小张", jobCenter);
    Experienced *wang = new Experienced("小王", jobCenter);
    Student *li = new Student("小李", jobCenter);
```

```
    // 求职中心发布职位需求信息，并通知全体求职者
    jobCenter->publishMessage("腾辉公司需要10个C++程序员。");
    jobCenter->notifyObservers();
    cout << endl;

    jobCenter->publishMessage("海景公司需要8个动画设计师。");
    jobCenter->notifyObservers();
    cout << endl;

    //发布一条重复信息
    jobCenter->publishMessage("海景公司需要8个动画设计师。");
    jobCenter->notifyObservers();
    cout << endl;

    jobCenter->publishMessage("仁海公司需要9个电工。");
    jobCenter->notifyObservers();
    cout << endl;

    delete li;
    ......  // 其他代码，完成收尾工作
}
```

9．程序执行完毕后，命令行窗口显示的结果：

```
求职中心成立了

发布招聘信息：腾辉公司需要 10 个 C++程序员。

通知所有求职者 ...
我叫小张，我是一名毕业生，我收到了招聘信息：腾辉公司需要 10 个 C++程序员。
我叫小王，我有工作经验，待遇要好。我收到了招聘信息：腾辉公司需要 10 个 C++程序员。
我叫小李，我是一名毕业生，我收到了招聘信息：腾辉公司需要 10 个 C++程序员。

发布招聘信息：海景公司需要 8 个动画设计师。

通知所有求职者 ...
我叫小张，我是一名毕业生，我收到了招聘信息：海景公司需要 8 个动画设计师。
我叫小王，我有工作经验，待遇要好。我收到了招聘信息：海景公司需要 8 个动画设计师。
我叫小李，我是一名毕业生，我收到了招聘信息：海景公司需要 8 个动画设计师。

该信息已经发布过了

发布招聘信息：仁海公司需要 9 个电工。

通知所有求职者 ...
我叫小张，我是一名毕业生，我收到了招聘信息：仁海公司需要 9 个电工。
我叫小王，我有工作经验，待遇要好。我收到了招聘信息：仁海公司需要 9 个电工。
我叫小李，我是一名毕业生，我收到了招聘信息：仁海公司需要 9 个电工。

求职中心关闭了
```