Practice of Programming
Q&A

Haopeng Chen

***RE**liable, **IN**telligent and **S**calable Systems Group (**REINS**)*
Shanghai Jiao Tong University
Shanghai, China
e-mail: chen-hp@sjtu.edu.cn

工程

- 建议用书上现成的工程，在其基础上修改和添加自己的行为

```
class EventWindow :public Window{
   public:
        EventWindow(Point xy,int w,int h,const string& title);
};
EventWindow::EventWindow(Point xy,int w,int h,const
string&title):Window(xy,w,h,title){}
```

报错是1>c:\users\d\documents\visual studio

报错是1>2010\projects\game\game\window.h(58): error C2248:

报错是1>“Fl_Window::Fl_Window”: 无法访问 private 成员(在“Fl_Window”类中声明)

注意变量作用域

- 建议用书上现成的工程，在其基础上修改和添加自己的行为

```
void draw_poly(Simple_window &win)
{
        Graph_lib::Polygon poly;              // make a shape (a polygon)
        poly.add(Point(300,200));    // add a point
        poly.add(Point(350,100));    // add another point
        poly.add(Point(400,200));    // add a third point
        poly.set_color(Color::red);  // adjust properties of poly
        win.attach (poly);          // connect poly to the window
}


int main()
{
        Point tl(100,100);          // to become top left  corner of window
        Simple_window win(tl,600,400,"Canvas");    // make a simple window
        win.wait_for_button();       // give control to the display engine
}
```

静态函数是类函数


```
struct Lines_window : Window {
    Lines_window(Point xy, int w, int h, const string& title );
    Open_polyline lines;
private:
    Button next_button;       // add (next_x,next_y) to lines
    Button quit_button;
    In_box next_x;
    In_box next_y;
    Out_box xy_out;

    static void cb_next(Address, Address); // callback for next_button
    void next();
    static void cb_quit(Address, Address); // callback for quit_button
    void quit();
};
```

静态函数是类函数


```
void Lines_window::cb_quit(Address, Address pw)
{  reference_to<Lines_window>(pw).quit(); }

void Lines_window::quit()
{    hide();}

void Lines_window::cb_next(Address, Address pw) {
    reference_to<Lines_window>(pw).next();
    cout << next_x.get_int();
    cout << next_y.get_int();
}

void Lines_window::next(){
    int x = next_x.get_int();
    int y = next_y.get_int();
    ……..
}
```

利用宏解决函数重复定义

- 条件编译宏

```
#ifndef <标识>
#define <标识>
.....
.....
#endif
```

- <标识>可以是自由命名的，但每个头文件的这个“标识”都应该是唯一的
- 如果<标识>没有定义，则编译#define后面的语句，否则就不再编译了

Practice of Programming 3 Implementation

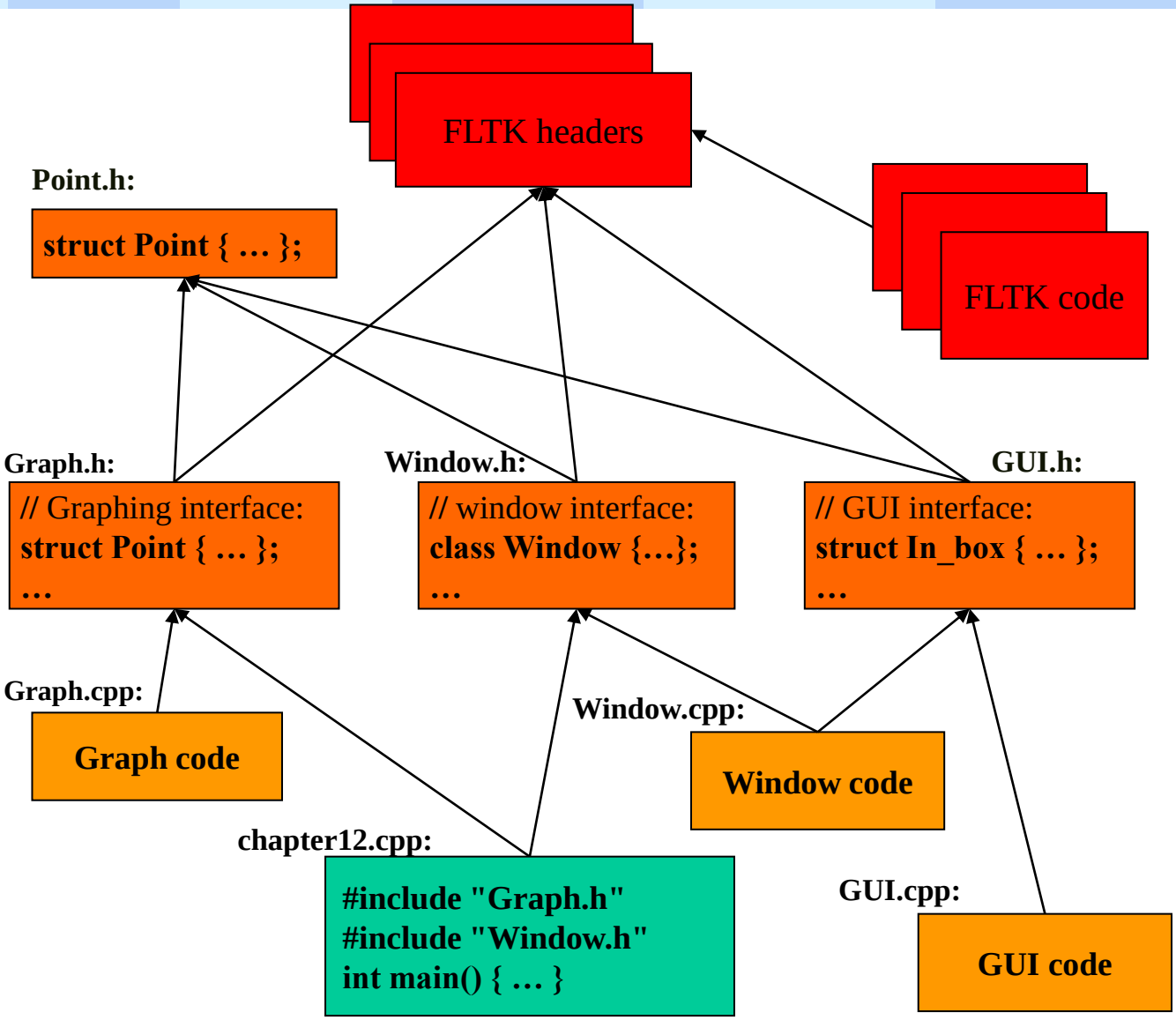
Haopeng Chen

***RE**liable, **IN**telligent and **S**calable Systems Group (**REINS**)*
Shanghai Jiao Tong University
Shanghai, China
e-mail: chen-hp@sjtu.edu.cn

内容提纲

- 回顾
  - 接口与实现分离
  - 纯虚函数
- 设计实例

代码组织


FLTK headers
FLTK code
Point.h:
struct Point { … };
Graph.h:
// Graphing interface:
struct Point { … };
…
Window.h:
// window interface:
class Window {…};
…
GUI.h:
// GUI interface:
struct In_box { … };
…
Graph.cpp:
Graph code
Window.cpp:
Window code
chapter12.cpp:
#include "Graph.h"
#include "Window.h"
int main() { … }
GUI.cpp:
GUI code

接口与实现分离

- 头文件 Header
  - 包含所有接口信息的文件(声明)
  - #include 到用户代码和实现代码中
- .cpp (“代码文件” / “实现文件”)
  - 包含用于实现在头文件中定义的接口的代码和使用这些接口的代码
  - #include 头文件
- 先阅读Graph.h头文件 – 接口
  - 然后再阅读 Graph.cpp 实现文件 – 实现
- Window.h头文件和window.cpp实现文件是不用阅读的
  - 当然，有些同学会看一眼
  - 当心：里面大量使用了目前还未讲解到的C++ 特性

类机制的问题

- 类机制，就像我们前面使用过的各种类一样，有一个缺点：
  - 它在接口(interface)的定义中
  - 混杂了一些实现(implementation)细节
- 类的实现包括：
  - 类的数据成员
  - 类的方法实现
- 方法实现可以从类的定义中分离
  - 进行单独编写(就像我们前面的示例代码一样)

类机制的问题

- 例如，前面我们开发的IntSet
- 就将其分成了两个文件：
  - intset.h --- IntSet类的定义
  - intset.cpp --- IntSet中每个方法的实现

- 遗憾的是，数据成员必须是类定义的一部分
- 由于使用IntSet类的所有程序员都必须能看到其定义
  - 因此这些程序员就知道了其部分实现

类机制的问题

- 在类定义中包含这些内容有两个副作用
  - 它使类定义变得复杂了
    - 使得它较难被阅读和理解
    - 考虑一下使用数组的IntSet实现
  - 它将部分信息暴露给了程序员
    - 而这些内容不应该暴露给程序员

- 第二个问题会产生非常严重的后果

类机制的问题

- 例如：
- 如果使用这个类的程序员错误地
  - 做出假设，认为我们的实现应该会提供某种“操作”
  - 但是这个类的接口并没有做出这样的承诺
- 那么我们就会陷入麻烦之中

- 我们有什么办法吗？

抽象基类 abstract base class

- 我们希望提供这样的类定义：
  - 不会将任何实现细节暴露给客户端程序员
  - 只包含接口信息
- 但是，因为类必须包括其数据成员
  - 因此，这意味着类根本就不能有任何实现

- 这没问题，因为我们可以使用纯接口类作为基类
  - 具体实现可以从这个类中导出
- 这样的基类被称为抽象基类
  - 或者有时被成为虚基类(virtual base class)
  - 因为我们利用了虚方法实现这一目的

抽象基类 abstract base class

- 下面是其具体工作机制
- 首先，要提供一个纯接口的IntSet定义
- 这个类永远都不会被实例化
  - 因为没有任何实现与其关联

- 因为这些方法没有任何实现
  - 所以我们要以特殊的方式声明它们
- 首先，我们将它们都声明成为virtual方法
- 其次，为这些虚方法赋值0

抽象基类 abstract base class


```
class IntSet {
   // OVERVIEW: a mutable set of bounded size containing integers
    public:
            virtual void insert(int v) = 0;
            // MODIFIES: this
            // EFFECTS: set = set + {v} if room, throws IntSetFull otherwise
            virtual void remove(int v) = 0;
            // MODIFIES: this
            // EFFECTS: set = set - {v}
            virtual bool query(int v) = 0;
            // EFFECTS: returns true if v is in set, false otherwise
                                        // EFFECTS: returns |set|
   };
```

纯虚函数

- 这些函数被称为纯虚函数(pure virtual function)
- 我们在它们还不存在的情况下提前进行了声明
- 可以这样考虑它们：它们是一组函数指针
  - 每个指针都指向NULL
- 注意：纯虚函数与虚函数并不一样

- 纯虚函数不需要任何定义
  - 强制所有的导出类去定义“它们自己的”版本

抽象类

- 包含一个或多个纯虚函数的类就是抽象类(abstract class)
- 我们不能创建抽象类的实例
  - 因为它还没有任何实现
- 例如，下面的代码会导致编译错误：

- 但是，可以定义指向抽象类的指针和引用
- 因此，下面两个变量声明都是合法的：

- 当然，如果没有IntSet的导出类提供实际的实现
  - 这些变量也就没什么实际用处

抽象类

- 具体实现可以由导出类提供：

```
const int MAXELTS = 100;
class IntSetImpl : public IntSet {
        int elts[MAXELTS];
        int numElts;
public:
        IntSetImpl();
        void insert(int v);
        void remove(int v);
        bool query(int v);
        int size();
};
```

抽象类

- 典型地，接口在公共头文件(*.h)中定义
  - 客户端可以引入(include)这个文件
- 然后，实现在源文件(*.cpp)中定义
  - 客户端只能链接(link)这个文件

- 因此，IntSet抽象类的客户端永远都不能看到IntSetImpl类的定义

抽象类

- 剩下唯一的事情是
  - 向客户端提供创建新的IntSet的方式
- 他们不能像通常方式那样做
  - IntSet s;
- 因为他们不能创建抽象类的实例
- 但是，他们也不能创建导出类的实例
  - 因为导出类的定义对他们是不可视的

- 我们怎么办？

抽象类

- 如果某个特定的类只有一个实例，那就有办法了
- .h 文件可以包含下面的访问函数原型：

```
IntSet *getIntSet();
// EFFECTS: returns a pointer to the IntSet
```

- 源文件可以定义其实现的单个静态的实例
  - 以及访问函数的函数体

```
static IntSetImpl i;
IntSet *getIntSet() { return &i; }
```

- 如果可以有多于一个的实例，那么就需要提供动态创建它们的函数

抽象类

- .h 文件中的函数要定义成static的：

```
static IntSet *getIntSet();
// EFFECTS: returns a pointer to the IntSet
```

- Static作用于类的成员
  - 数据成员和函数成员
  - 表示这些成员不属于某个特有的对象
  - 而是所有这个类的对象都具有的属性
  - 因此，不需要创建任何对象，即可在类上调用
  - IntSet* s = IntSet::*getIntSet()*

要点

- 抽象类是不完整的类
  - 它可以包含不是纯虚函数的成员函数
  - 只能当做基类使用
  - 不能从中创建任何对象
    - 因为它没有所有成员的完整定义
- 抽象类的导出类自动是抽象类
  - 除非它提供了所有纯虚函数的定义
  - 并且不会引入新的纯虚函数

总结

- 延迟绑定将调用哪个函数实现的决定延迟到了运行时
- 纯虚函数没有任何定义
  - 具有至少一个纯虚函数的类是抽象类
  - 从抽象类中不能创建任何对象
  - 其用法严格限制为只能作为基类供其他类导出
- 导出类对象可以赋值给基类对象
  - 成员会丢失，切片问题
- 指针赋值和动态对象
  - 可以修复切片问题
- 将所有析构器都设成是virtual的
  - 确保内存会被正确释放

内容提纲

- 继承与多态
- 回顾
  - 接口与实现分离
  - 纯虚函数
- 设计原则

内容提纲

- 回顾
  - 接口与实现分离
  - 纯虚函数
- 设计实例

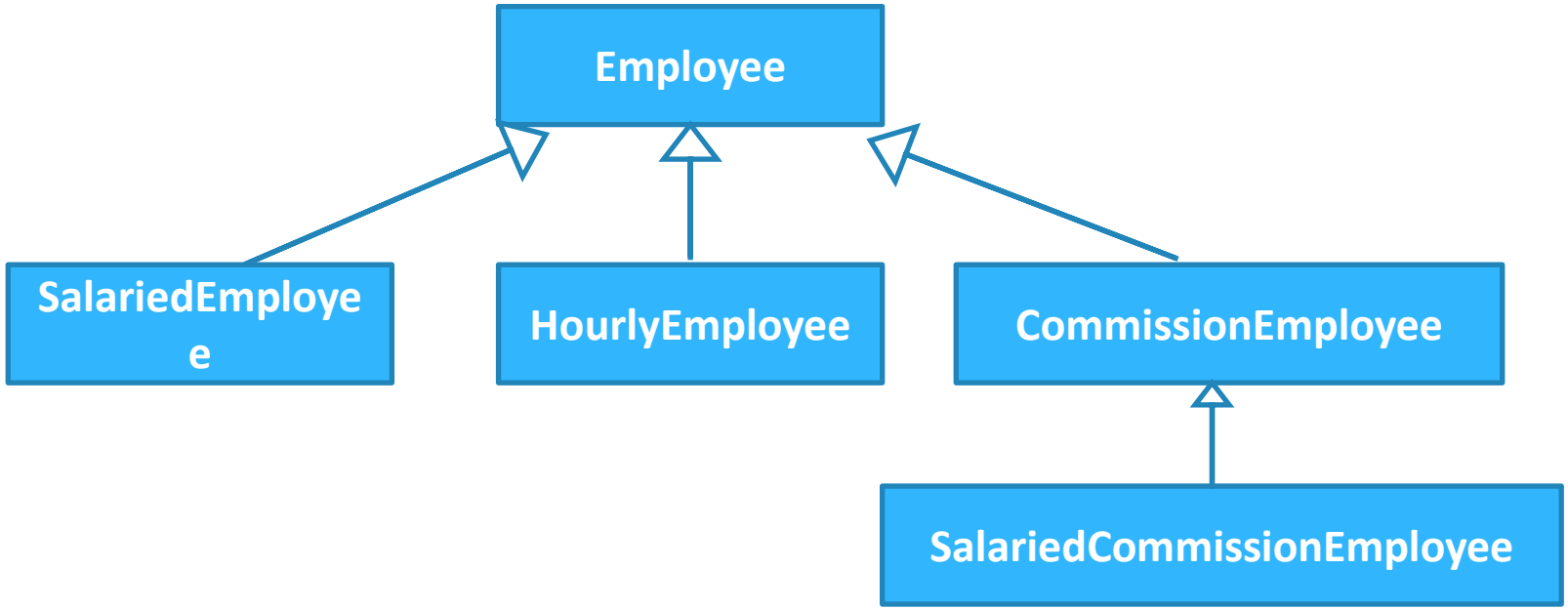
问题描述

- 企业中有4类员工，薪酬支付方式不相同

	earnings	print
Employee	0	*firstName lastName* social security number: *ssn*
Salaried Employee	weeklySalary	salaried employee: *firstName lastName* social security number: *ssn* weekly salary: *weeklySalary*
Hourly Employee	If hours<=40 wage * hours Else (40*wage)+ ((hours-40)*wage*1.5)	hourly employee: *firstName lastName* social security number: *ssn* hourly wage: *wage*; hours worked: *hours*
Commission Employee	commissionRate* grossSales	commission employee: *firstName lastName* social security number: *ssn* gross sales: *grossSales*; commission rate: *commissionRate*
BasePlus Commission Employee	baseSalary + commissionRate* grossSales	Base salaried commission employee: *firstName lastName* social security number: *ssn* gross sales: *grossSales*; commission rate: *commissionRate* base salary: *baseSalary*

问题描述

- 企业中有4类员工，薪酬支付方式不相同



Thank You!