ГОСУДАРСТВЕННЫЙ КОМИТЕТ СВЯЗИ, ИНФОРМАТИЗАЦИИ
И ТЕЛЕКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ РЕСПУБЛИКИ
УЗБЕКИСТАН
ТАШКЕНТСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ
ТЕХНОЛОГИЙ

Кафедра: «ПОИТ»

Курсовая работа

По предмету:

«Объектно – ориентированный языки программирования»

Выполнила: студентка гр223-10ИТр

Кучимова Санобар

Принял(а): _____

Ташкент - 2013

«Утверждаю»

Заведующий кафедрой

_____

«__»_____201_г.

Задание

На курсовую работу

По дисциплине: Объектно-ориентированные языки программирования

Студентка : Кучимова Санобар группы 223-10ИТр

Руководитель: _____

Защита курсовой работы: _____

Тема: Классы (тема №2). Номер варианта 2.20

Задание: Определить иерархию объектов (в соответствии с вариантом). Реализовать путем наследования методы для автоматизации поиска указанной информации и указанных операций.

Вариант **2.20.** АПТЕКА (название, ОТДЕЛЫ, ЛЕКАРСТВА, СОТРУДНИКИ)

Все данные относительно лекарства

Содержание работы. Объем __ листов А4.

Подпись преподавателя: _____

Подпись студента: _____

Дата выдачи задания:

Аннотация

Данная курсовая работа по предмету: «Объектно-ориентированные языки программирования» на тему: «Отношения между классами (тема №2). Номер варианта 2.20». В данной курсовой работе рассматриваются иерархия объектов (в соответствии с вариантом). Реализованные путем наследования методы для автоматизации поиска указанной информации и указанных операций – Вывод данных относительно Лекарства. Программа разработана на языке программирования C++, в среде Visual Studio 2010.

Объем работы ____ листов

Содержание

Цель

Создание иерархии объектов.

Постановка задачи

Определить иерархию объектов (в соответствии с вариантом). Реализовать путем наследования методы для автоматизации поиска указанной информации и указанных операций.

Вариант задания

Вариант **2.20.** АПТЕКА (название, ОТДЕЛЫ, ЛЕКАРСТВА, СОТРУДНИКИ)
Все данные относительно лекарства

Реализация задания

Теоретическая часть

Насле́дование — механизм объектно-ориентированного программирования (наряду с инкапсуляцией, полиморфизмом и абстракцией), позволяющий описать новый класс на основе уже существующего (родительского), при этом свойства и функциональность родительского класса заимствуются новым классом.

Другими словами, класс-наследник реализует спецификацию уже существующего класса (базовый класс). Это позволяет обращаться с объектами класса-наследника точно так же, как с объектами базового класса...

Класс, от которого произошло наследование, называется базовым или родительским (англ. base class). Классы, которые произошли от базового, называются потомками, наследниками или производными классами (англ. derived class).

В некоторых языках используются абстрактные классы. Абстрактный класс — это класс, содержащий хотя бы один абстрактный метод, он описан в программе, имеет поля, методы и не может использоваться для непосредственного создания объекта. То есть от абстрактного класса можно только наследовать. Объекты создаются только на основе производных классов, наследованных от абстрактного. Например, абстрактным классом может быть базовый класс «сотрудник вуза», от которого наследуются классы «аспирант», «профессор» и т. д. Так как производные классы имеют общие поля и функции (например, поле «год рождения»), то эти члены класса могут быть описаны в базовом классе. В программе создаются объекты на основе классов «аспирант», «профессор», но нет смысла создавать объект на основе класса «сотрудник вуза».

Множественное наследование

При множественном наследовании у класса может быть более одного предка. В этом случае класс наследует методы всех предков. Достоинства такого подхода в большей гибкости. Множественное наследование реализовано в C++. Из других языков, предоставляющих эту возможность, можно отметить Python и Эйфель. Множественное наследование поддерживается в языке UML.

Множественное наследование — потенциальный источник ошибок, которые могут возникнуть из-за наличия одинаковых имен методов в предках. В языках, которые позиционируются как наследники C++ (Java, C# и др.), от множественного наследования было решено отказаться в пользу интерфейсов. Практически всегда можно обойтись без использования данного механизма. Однако, если такая необходимость все-таки возникла, то, для разрешения конфликтов использования наследованных методов с одинаковыми именами, возможно, например, применить операцию расширения видимости — «::» — для вызова конкретного метода конкретного родителя.

Попытка решения проблемы наличия одинаковых имен методов в предках была предпринята в языке Эйфель, в котором при описании нового класса необходимо явно указывать импортируемые члены каждого из наследуемых классов и их именование в дочернем классе.

Большинство современных объектно-ориентированных языков программирования (C#, Java, Delphi и др.) поддерживают возможность одновременно наследоваться от класса-предка и реализовать методы нескольких интерфейсов одним и тем же классом. Этот механизм позволяет во многом заменить множественное наследование — методы интерфейсов необходимо переопределять явно, что исключает ошибки при наследовании функциональности одинаковых методов различных классов-предков.

Наследование дает возможность объявить производный класс, который наследует свойства, данные, методы и события всех своих предшественников в иерархии классов, а также может объявлять новые характеристики и перегружать некоторые из наследуемых функций.

Обобщенный синтаксис объявления производного класса:

class <имя класса>: <область видимости>

<имя родительского класса> {…}

Наследование конструкторов

Поскольку конструкторы не наследуются, при создании производного класса наследуемые им данные-члены должны инициализироваться конструктором базового класса. Конструктор базового класса вызывается автоматически и выполняется до конструктора производного класса.

Виртуальные методы

К механизму виртуальных функций обращаются в тех случаях, когда в каждом производном классе требуется свой вариант некоторой компонентной функции. Классы, включающие такие функции, называются **полиморфными** и играют особую роль в ООП.

Виртуальные функции предоставляют механизм **позднего** (**отложенного**) или **динамического связывания.**

При позднем связывании адреса определяются динамически во время выполнения программы, а не статически во время компиляции, как в традиционных компилируемых языках, в которых применяется *раннее связывание*.

Виртуальность наследуется. После того как функция определена как виртуальная, ее повторное определение в производном классе (с тем же самым прототипом) создает в этом классе новую виртуальную функцию.

Все методы и переменные объектов могут быть замещены по умолчанию. Если же вы хотите объявить, что подклассы не имеют права замещать какие-либо переменные и методы вашего класса, вам нужно объявить их как final (в Delphi / C++ не писать слово **virtual**).

final int FILE_NEW = 1;

По общепринятому соглашению при выборе имен переменных типа final — используются только символы верхнего регистра (т.е. используются как аналог препроцерных констант C++). Использование final-методов порой приводит к выигрышу в скорости выполнения кода — поскольку они не могут быть замещены, транслятору ничто не мешает заменять их вызовы *встроенным* (in-line) кодом (байт-код копируется непосредственно в код вызывающего метода).

Реализация. Описание процедур

```
#include <fstream>
#include <string>
using namespace std;
ifstream cin ("input.in");
ifstream searcher ("z.in");
ofstream cout("output.out");
```

Класс лекарств

```
class Lekarstva
{
	public:
	string name;
	void Show_Lekarstva()
	{
		cout<<"Название лекарства: "<<name<<endl;
	}
	Lekarstva( string l_name)
	{
		name = l_name;
	}
	string get_lec()
	{
		return name;
	}
	Lekarstva() {}
	~Lekarstva() {}
};
```

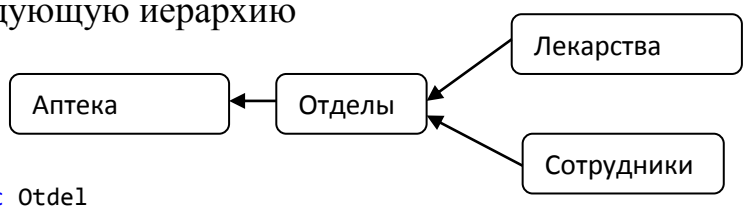
Класс сотрудников

```
class Sotrudniki
{
	public:
	string name;
	void Show_Sotrudniki()
	{
		cout<<"Имя сотрудника: "<< name <<endl;
	}
	Sotrudniki(string n)
	{
		name = n;
	}
	Sotrudniki()  {}
	~Sotrudniki() {}
};
```

Класс отделы, где двумя полями класса являются объекты классов лекарства и сотрудники

```
class Otdel
{
	public:
	string name1;
	int s,l;
	Lekarstva M;
	Sotrudniki F;
	int get_id()
	{
		return l;
	}
	string get_lec()
	{
		return M.get_lec();
	}
	void Show_Otdel()
	{
		cout<<"Название отдела: "<< name1 << " \nномер сотрудника: "<< s << "
идентификатор лекарства: " << l <<endl;
		F.Show_Sotrudniki();
		M.Show_Lekarstva();
	}
	Otdel(string otd_n, int num_s, int num_l, string s_name, string lec_name)
	{
		name1 = otd_n;
		s = num_s;
		l = num_l;
		F = Sotrudniki(s_name);
		M = Lekarstva(lec_name);
	}
	Otdel() {}
	~Otdel() {}
};
```

Класс Отдел наследует класс Аптека.
Имеем следующую иерархию




```
class Apteka: public Otdel
{
	public:
	string name1,fam,im,ot;
	int v_age;
	Otdel P;
	void Show_t_s()
	{
		cout<<"Аптека: "<< name1 <<"\nИмя владельца: " << fam << " " << im << " "
<< ot << " " << "возраст: "<< v_age << endl;
		P.Show_Otdel();
	}
	int get_id()
	{
		return P.get_id();
	}
	string get_lec()
	{
		return P.get_lec();
	}
	Apteka(string nazvanie, string f, string i, string o, int age, string otd_n, int
num_l, int num_s, string name, string lec_name)
```

```
	{
		name1 = nazvanie;
		fam = f;
		im = i;
		ot = o;
		v_age = age;
		P = Otdel(otd_n,num_l,num_s,name,lec_name);
	}
	Apteka() {}
	~Apteka() {}
};
```

Выводятся все данные относительно лекарства,

```
	void Show_t_s()
	{
		cout<<"Аптека: "<< name1 <<"\nИмя владельца: " << fam << " "
		    << im << " " << ot << " " << "возраст: "<< v_age << endl;
		P.Show_Otdel();
	}
```

Класс поисковик, который ищет название лекарства по его идентификационному номеру

```
class Searcher
{
	public:
	int m_id;
	void show_result(string  n)
	{
		cout<<"\nНазвание лекарства соответвующее заданному идентификатору: " << n
<<endl;
	}
	string Search (Apteka a[], int count)
	{
		int c=0;
		for(int i = 0;i<count;i++)
		{
			if(a[i].get_id() == m_id) return a[i].get_lec();
		}
		return NULL;
	}
	Searcher(int id)
	{
		m_id = id;
	}
	Searcher() {}
	~Searcher() {}
};
```

Запрашиваем у пользователя количество записей

```
	cin>>count;
```

Создаем массив объектов класса Аптека и объект класса поиска

```
Apteka *TS = new Apteka[count1];
Searcher *S = new Searcher[1];
```

и циклически забиваем массив, в конце цикла вызываем конструктор класса Аптека.

```
for(int i=0;i<count1;i++)
{
	cin>>nazvanie1>>f1>>i1>>o1>>age1>>otd_n1>>num_s1>>num_l1>>name1>>lec_name1;
	TS[i] = Apteka(nazvanie1, f1, i1, o1, age1, otd_n1, num_l1, num_s1, name1,
lec_name1);
	TS[i].Show_t_s(); cout<<"\n";
}
```

Вводимые данные в программу:

```
3
Привокзальная
Хван Дмитрий Артемович 22
отдел1 20 3
Андрей Иутирокс

Местная
Хван Дмитрий Артемович 22
отдел1 9 3
Галина Грипхот

При_поликлинике
Абдурахманов Артем Алишерович 22
отдел1 14 3
Вася Пупиндон
```

Вывод данных:

```
Аптека: Привокзальная
Имя владельца: Ли Алексей Артемович возраст: 22
Название отдела: отдел1
номер сотрудника: 3 идентификатор лекарства: 20
Имя сотрудника: Андрей
Название лекарства: Иутирокс

Аптека: Местная
Имя владельца: Хван Евгений Игоревич возраст: 22
Название отдела: отдел1
номер сотрудника: 3 идентификатор лекарства: 9
Имя сотрудника: Галина
Название лекарства: Грипхот

Аптека: При_поликлинике
Имя владельца: Кучимова Санобар Рахматовна возраст: 22
Название отдела: отдел1
номер сотрудника: 3 идентификатор лекарства: 14
Имя сотрудника: Вася
Название лекарства: Пупиндон


Название лекарства соответвующее заданному идентификатору: Грипхот
```

Файл запроса:

```
Идентификатор лекарства: 9
```

Полный исходный код

```
#include <fstream>
#include <string>
using namespace std;
ifstream cin ("input.in");
ifstream searchin ("zapros.in");
ofstream cout("output.out");

// ========================== класс Лекарства ============================== //
class Lekarstva
{
	public:
	string name;
	void Show_Lekarstva()
	{
		cout<<"Название лекарства: "<<name<<endl;
	}
	Lekarstva( string l_name)
	{
		name = l_name;
	}
	string get_lec()
	{
		return name;
	}
	Lekarstva() {}
	~Lekarstva() {}
};
// ========================== класс Лекарства ============================== //

// ============================ класс Сотрудников ================================ //
class Sotrudniki
{
	public:
	string name;
	void Show_Sotrudniki()
	{
		cout<<"Имя сотрудника: "<< name <<endl;
	}
	Sotrudniki(string n)
	{
		name = n;
	}
	Sotrudniki()  {}
	~Sotrudniki() {}
};
// ============================ класс Сотрудников ================================ //

// ============================ класс Отделов ================================== //
class Otdel
{
	public:
	string name1;
	int s,l;
	Lekarstva M;
	Sotrudniki F;
	int get_id()
	{
		return l;
	}
	string get_lec()
	{
		return M.get_lec();
```

```
	}
	void Show_Otdel()
	{
		cout<<"Название отдела: "<< name1 << " \nномер сотрудника: "<< s << "
идентификатор лекарства: " << l <<endl;
		F.Show_Sotrudniki();
		M.Show_Lekarstva();
	}
	Otdel(string otd_n, int num_s, int num_l, string s_name, string lec_name)
	{
		name1 = otd_n;
		s = num_s;
		l = num_l;
		F = Sotrudniki(s_name);
		M = Lekarstva(lec_name);
	}
	Otdel() {}
	~Otdel() {}
};
// ============================ класс Отделов ================================== //


// ====================== класс Аптека ============================== //
class Apteka: public Otdel
{
	public:
	string name1,fam,im,ot;
	int v_age;
	Otdel P;
	void Show_t_s()
	{
		cout<<"Аптека: "<< name1 <<"\nИмя владельца: " << fam << " " << im << " "
<< ot << " " << "возраст: "<< v_age << endl;
		P.Show_Otdel();
	}
	int get_id()
	{
		return P.get_id();
	}
	string get_lec()
	{
		return P.get_lec();
	}
	Apteka(string nazvanie, string f, string i, string o, int age, string otd_n, int
num_l, int num_s, string name, string lec_name)
	{
		name1 = nazvanie;
		fam = f;
		im = i;
		ot = o;
		v_age = age;
		P = Otdel(otd_n,num_l,num_s,name,lec_name);
	}
	Apteka() {}
	~Apteka() {}
};
// ====================== класс Аптека ============================== //


// ========================== класс поисковик ================================ //
class Searcher
{
	public:
	int m_id;
```

```
	void show_result(string  n)
	{
		cout<<"\nНазвание лекарства соответвующее заданному идентификатору: " << n
<<endl;
	}
	string Search (Apteka a[], int count)
	{
		int c=0;
		for(int i = 0;i<count;i++)
		{
			if(a[i].get_id() == m_id) return a[i].get_lec();
		}
		return NULL;
	}
	Searcher(int id)
	{
		m_id = id;
	}
	Searcher() {}
	~Searcher() {}
};
// ============================ класс поисковик ================================ //

int main()
{
	int age1,num_l1,num_s1,count1,id;
	string nazvanie1,f1,i1,o1,otd_n1,name1,lec_name1,x,result;
	cin>>count1;
	Apteka *TS = new Apteka[count1];
	Searcher *S = new Searcher[1];
	for(int i=0;i<count1;i++)
	{
		cin>>nazvanie1>>f1>>i1>>o1>>age1>>otd_n1>>num_s1>>num_l1>>name1>>lec_name1;
		TS[i] = Apteka(nazvanie1, f1, i1, o1, age1, otd_n1, num_l1, num_s1, name1,
lec_name1);
		TS[i].Show_t_s(); cout<<"\n";
	}
	// Выбрать название того лекарства которое соответствует идентификатору указанному
в запросе
	searchin>>x>>x>>id;
	S[0] = Searcher(id);
	result = S[0].Search(TS,count1);
	S[0].show_result(result);
	return 0;
}
```

Заключение

Данная курсовая работа по предмету: «Объектно-ориентированные языки программирования» на тему: «Отношения между классами (тема №2). Номер варианта 2.20». В данной курсовой работе были рассмотрены иерархия объектов. Реализация путем наследования методов для автоматизации поиска указанной информации, также реализован автоматический вывод всех данных относительно лекарства. Программа была реализована на C++, в среде visual Studio 2010.

Список используемой литературы

- ✓ http://msdn.microsoft.com
- ✓ http://forum.sources.ru/
- ✓ http://www.codeproject.com/
- ✓ http://www.smallguru.com/
- ✓ http://www.cyberforum.ru/