Лекция. Чистые виртуальные функции. Абстрактные классы.

Чистые виртуальные функции

Иногда вам может понадобиться включить в базовый класс виртуальную функ­цию — так, чтобы она была переопределена в производном классе в соответствии с назначением этого класса, но в базовом классе для нее пока нет осмысленного опре­деления.

Например, вы предположительно можете иметь класс *CContainer*, который мо­жет служить базовым для определения класса *СВох*, или класса *CBottle*, или даже класса *CTeapot*. Класс *CContainer* не будет иметь полей, но вы можете решить все-таки предоставить виртуальную функцию-член *Volume ()* для любого производного класса. Поскольку класс *CContainer* не имеет полей, а, сле­довательно, не имеет размеров, то и не может быть осмысленного определения функ­ции *Volume ().* Однако вы можете определить класс, включающий метод *Volume (),* следующим образом:

*#include <iostream.h>*

*class CContainer //Общий базовый класс для специфических контейнеров*

*{*

*public:*

***//Функция для вычисления объема - без содержимого***

***//Это определение чистой ('pure') виртуальной функции***

***virtual double Volume() const = 0;***

*//Функция для отображения объема*

*virtual void ShowVolume() const*

*{ cout << endl*

*<<"Volume equals" << Volume();*

*}*

*};*

Оператор для виртуальной функции *Volume ()* определяет ее, как не имеющую содержимого, путем размещения знака равно и нуля в заголовке. Это называется чистой (*рurе*) виртуальной функцией. Любой класс, унаследованный от данного, должен либо определять функции *Volume ()*, либо переопределять ее повторно как чистую виртуальную. Поскольку вы объявили Volume () как *const*, ее реализация в любом производном классе также должна быть *const*.

Вспомните, что вариации const и не-const функции с одинаковыми именами и списками параметров являются разными функциями. Другими словами, функцию можно перегрузить, добавив *const*.

Класс также включает функцию *ShowVolume ()*, которая отображает объем объектов производных классов. Поскольку она объявлена как *virtual*, то может быть заменена в производном классе, но если нет, будет вызвана ее версия из базового класса, которую вы видите здесь

**Абстрактные классы**

Класс, содержащий чистую виртуальную функцию, называется абстрактным классом. Он называется абстрактным, потому что определить объекты класса, включающего чистую виртуальную функцию, невозможно. Такой класс существует только для того, чтобы можно было определять классы, производные от него. Если

класс, производный от абстрактного класса, также определяет чистую виртуальную функцию, он также является абстрактным.

Из предыдущего примера класса CContainer вы не должны заключить, что аб­страктный класс не может иметь полей. Наличие чистой виртуальной функции - это единственное условие, которое делает класс абстрактным.

Иногда абстрактный класс может иметь более одной чистой виртуальной функции. Тогда производный от него класс должен реализовать каждую из этих функций, иначе он тоже будет абстрактным. Если вы забудете объявить версию функции Volume () в производном классе как const, то производный класс останется абстрактным, потому что он будет включать чистый виртуальный метод *Volume ()*, которая является const, наряду с He-*const*-функцией *Volume (),* которую вы определите.

Вы можете реализовать класс *ССаn*, представляющий банки пива или колы, вместе с оригинальным классом *СВох*, как наследников класса *CContainer*.

Определение класса *СВох* как подкласса *CContainer* будет таким:

*class CBox : public CContainer //Производный класс*

*{*

*public:*

*//Функция для отображения объема объекта*

*virtual void ShowVolume() const*

*{ cout << endl*

*<<"Volume CBox equals" << Volume();*

*}*

*//Функция для вычисления объекта CBox*

*virtual double Volume() const*

*{return m\_Length\*m\_Width\*m\_Height;}*

*//Конструктор*

*CBox (double lv = 1.0,double wv = 1.0, double hv = 1.0)*

*:m\_Length(lv), m\_Width (wv), m\_Height(hv) {}*

*protected:*

*double m\_Length;*

*double m\_Width;*

*double m\_Height;*

*};*

Класс *СВох*, по сути, тот же, что и в предыдущем примере, за исключением того, что теперь он объявлен как производный от класса *CContainer*. Функция *Volume ()* полностью определена внутри класса (как и должно быть, если этот класс будет служить для создания объектов). Единственный альтернативный вариант заключается в спецификации ее как чистой виртуальной функции, поскольку она является таковой в базовом классе, но тогда мы не смогли бы создавать объекты *СВох*.

Класс *ССаn* вы можете определить следующим образом:

*class CCan : public CContainer*

*{*

*public:*

*//Функция вычисления объема банки*

*virtual double Volume() const*

*{return 0.25\*PI\*m\_Diameter\*m\_Diameter\*m\_Height;}*

*//Конструктор*

*CCan (double hv = 4.0, double dv = 2.0)*

*:m\_Height(hv), m\_Diameter(dv) {}*

*protected:*

*double m\_Height;*

*double m\_Diameter;*

*};*

В классе *CCan* также определена функция *Volume ()* на основании формулы , где h - высота банки, а г- радиус ее поперечного сечения. Объем вычисляется как высота, умноженная на площадь сечения. Выражение в определении функции

предполагает, что определена глобальная константа *PI*, поэтому мы имеем внешний оператор, указывающий, что *PI* - глобальная переменная типа const double. Также обратите внимание, что мы переопределили функцию *ShowVolume ()* в классе *СВох*, но не в классе *ССаn*. Вы увидите эффект от этого, когда мы получим результат работы программы.

//listing 56

***//Чистые виртуальные функции***

*const double PI = 3.14159265; //Глобальное определение PI*

*#include <iostream.h>*

*class CContainer //Общий базовый класс для специфических контейнеров*

*{*

*public:*

***//Функция для вычисления объема - без содержимого***

***//Это определение чистой ('pure') виртуальной функции, на***

***//что указывает '=0'***

***virtual double Volume() const = 0;***

*//Функция для отображения объема*

*virtual void ShowVolume() const*

*{*

*cout << endl*

*<<"Volume equals" << Volume();*

*}*

*};*

*class CBox : public CContainer //Производный класс*

*{*

*public:*

*//Функция для отображения объема объекта*

*virtual void ShowVolume() const*

*{ cout << endl*

*<<"Volume CBox equals" << Volume();*

*}*

*//Функция для вычисления объема объекта CBox*

*virtual double Volume() const*

*{return m\_Length\*m\_Width\*m\_Height;}*

*//Конструктор*

*CBox (double lv = 1.0,double wv = 1.0, double hv = 1.0)*

*:m\_Length(lv), m\_Width (wv), m\_Height(hv) {}*

*protected:*

*double m\_Length;*

*double m\_Width;*

*double m\_Height;*

*};*

*class CCan : public CContainer*

*{*

*public:*

*//Функция вычисления объема банки*

*virtual double Volume() const*

*{return 0.25\*PI\*m\_Diameter\*m\_Diameter\*m\_Height;}*

*//Конструктор*

*CCan (double hv = 4.0, double dv = 2.0)*

*:m\_Height(hv), m\_Diameter(dv) {}*

*protected:*

*double m\_Height;*

*double m\_Diameter;*

*};*

*int main(void)*

*{*

*//указатель на абстрактный базовый класс,*

*//инициализированный адресом объекта CBox*

*CContainer\* pC1 = new CBox(2.0, 3.0, 4.0);*

*//указатель на абстрактный базовый класс,*

*//инициализированный адресом объекта CCan*

*CContainer\* pC2 = new CCan(6.5, 3.0);*

*pC1->ShowVolume(); //вывод объемов двух объектов через указатели*

*pC2->ShowVolume();*

*cout << endl;*

*delete pC1; //освободить память в свободном хранилище*

*delete pC2;*

*return 0;*

*}*

*Volume CBox equals24*

*Volume equals45.9458*

В этой программе объявлены два указателя на базовый класс *CContainer*. Хотя вы не можете объявлять объекты *CContainer* (поскольку *CContainer -* абстрактный класс), все же вы можете определить указатель на CContainer и затем использовать его для хранения адреса объекта производного класса; фактически он может служить для хранения адреса любого объекта, тип которого является прямым или непрямым наследником *CContainer*. Указателю *рС1* присвоен адрес объекта *СВох*, созданного в свободном хранилище с помощью операции new. Второму указателю подобным же образом присвоен адрес объекта *ССаn*.

Конечно, поскольку объекты производных классов созданы динамически, вы должны выпол­нить операцию delete, чтобы очистить место в свободном хранилище, когда необходимость в этих объектах отпадет.

Поскольку функция *ShowVolume ()* определена в классе *СВох*, именно версия этой функции из производного класса вызывается для объекта *СВох*. Эта функция не определена в классе *ССаn*, поэтому для объекта *ССаn* вызывается версия этой функции из базового класса, унаследованная *ССаn*. Поскольку *Volume ()* - виртуальная функция, реализованная в обоих производных классах (что необходимо, потому что в базовом классе она является чистой виртуальной), ее вызов разрешается при выполнении программы выбором версии, относящейся к классу объекта, на который установлен указатель. Поэтому для указателя *рС1*

вызывается версия из класса *СВох*, а для указателя *рС2* - версия из класса *ССаn*. В каждом случае, таким образом, получается корректный результат.

Точно так же вы могли бы использовать только один указатель, и присвоить ему адрес объекта *ССаn* (после вызова функции *Volume ()* объекта *СВох*). Указатель на базовый класс может содержать адрес объекта любого производного класса, даже когда несколько различных классов унаследованы от одного и того же базового класса, и потому вы можете иметь автоматический выбор правильной виртуальной функции для всего диапазона производных классов. Впечатляет, не правда ли?

**Непрямые базовые классы**

Ранее упоминалось, что базовый класс подкласса, в свою очередь, может быть унаследован от другого, «еще более базового» класса. Маленькое расширение последнего примера предлагает иллюстрацию этого, а также демонстрацию ис­пользования виртуальной функции на втором уровне наследования.

Все, что потребуется сделать - это добавить класс *CGlassBox* к классам, перечисленным в предыдущем примере. Отношения между классами показаны на рисунке.



Класс *CGlassBox*, как и ранее, унаследован от *СВох*, но мы опустили версию *ShowVolume ()* производного класса, чтобы продемонстрировать, что версия базо­вого класса по-прежнему распространяется на производные классы. В показанной здесь иерархии классов класс *CContainer* является непрямым базовым классом для *CGlassBox* и прямым базовым - для *СВох* и *ССаn*.

//listing 57

*//непрямые базовые классы*

*const double PI = 3.14159265; //Глобальное определение PI*

*#include <iostream.h>*

*class CContainer //Общий базовый класс для специфических контейнеров*

*{*

*public:*

*//Функция для вычисления объема - без содержимого*

*//Это определение чистой ('pure') виртуальной функции, на что //указывает '=0'*

*virtual double Volume() const = 0;*

*//Функция для отображения объема*

*virtual void ShowVolume() const*

*{ cout << endl*

*<<"Volume equals" << Volume();*

*}*

*};*

*class CBox : public CContainer //Производный класс*

*{*

*public:*

*//Функция для отображения объема объекта*

*virtual void ShowVolume() const*

*{ cout << endl*

*<<"Volume CBox equals" << Volume();*

*}*

*//Функция для вычисления объекта CBox*

*virtual double Volume() const*

*{return m\_Length\*m\_Width\*m\_Height;}*

*//Конструктор*

*CBox (double lv = 1.0,double wv = 1.0, double hv = 1.0)*

*:m\_Length(lv), m\_Width (wv), m\_Height(hv) {}*

*protected:*

*double m\_Length;*

*double m\_Width;*

*double m\_Height;*

*};*

*class CCan : public CContainer*

*{*

*public:*

*//Функция вычисления объема банки*

*virtual double Volume() const*

*{return 0.25\*PI\*m\_Diameter\* m\_Diameter\*m\_Height;}*

*//Конструктор*

*CCan (double hv = 4.0, double dv = 2.0)*

*:m\_Height(hv), m\_Diameter(dv) {}*

*protected:*

*double m\_Height;*

*double m\_Diameter;*

*};*

*class CGlassBox : public CBox //Производный класс*

*{*

*public:*

*//функция производного класса для вычисления объема CGlassBox //резервирующая 15% на упаковку*

*virtual double Volume()const*

*{ return 0.85\*m\_Length\* m\_Width\* m\_Height; }*

*//Конструктор*

*CGlassBox (double lv, double wv, double hv):CBox(lv, wv, hv) { }*

*}*

*int main(void)*

*{*

*//указатель на абстрактный базовый класс,*

*//инициализированный адресом объекта CBox*

*CContainer\* pC1 = new CBox(2.0, 3.0, 4.0);*

*CCan myCan(6.5, 3.0); //Определение объекта CCan*

*CGlassBox myGlassBox(2.0, 3.0, 4.0); //Определение объекта //CGlassBox*

*pC1->ShowVolume(); //вывод объема CBox*

*delete pC1;*

*//инициализация адресом объекта CCan*

*pC1 = &myCan; //присвоить указателю адрес myCan*

*pC1->ShowVolume(); //вывод объема CCan*

*pC1 = &myGlassBox; //присвоить указателю адрес myGlassBox*

*pC1->ShowVolume(); //вывод объема CGlassBox*

*cout << endl;*

*return 0;*

*}*

*Результат*

*Volume CBox equals24*

*Volume equals45.9458*

*Volume CBox equals20.4*

В этом примере была реализавана трехуровневая иерархия классов, показанная на рисунке, с *CContainer* в качестве абстрактного базового класса, поскольку он содержит чистую виртуальную функцию *Volume ().* Функция *main ()* теперь вызывает функцию *ShowVolume ()* три раза, используя один и тот же указатель базового класса, но при этом указатель каждый раз содержит адреса объектов разных классов. Поскольку *ShowVolume ()* не определена ни в одном из производных классов, каждый раз вызывается ее версия из базового класса. Отдельная ветвь от базы CContainer определяет производный класс *ССаn*.

Вывод показывает, что в соответствии с типом объекта, каждый раз для выполнения выбирается одна из трех разные версий *Volume ()*.

Обратите внимание, что вы должны удалять объект *СВох* из свободного хранилища перед тем, как присваивать указателю другое значение адреса. Если этого не делать, вы не сможете очистить память, выделенную в свободном хранилище, потому что адрес оригинального объекта не сохранится. Это ошибка, которую легко допустить при переназначении указателей и использовании свободного хранилища.