ООП. Инкапсуляция, классы и объекты

Инкапсуляция (encapsulation, incapsulation) – это способ формирования объектов в ООП, состоящий в ограничении внешнего доступа к данным и объединении данных с методами (алгоритмами их обработки).

Этот механизм, связывает воедино код и данные, которыми код манипулирует, и защищает их от несанкционированного и неправильного использования. В ООЯ код и данные можно "упаковать" в "чёрный ящик" – **объект (object)**. Объект и есть средство инкапсуляции.

Объект представляет собой сложную переменную, тип которой определён программистом.

Внутри объекта код и данные могут быть закрытыми (private) или открытыми (public). Открытая часть объекта доступна извне из любой части программы и обеспечивает управляемое взаимодействие (интерфейс) между объектами.

Основным механизмом инкапсуляции является логическая абстракция **класс** – модель реального объекта, **объект**, в свою очередь, это конкретный экземпляр класса, его физическая реализация.

Класс – это тип данных, определяемый программистом, в котором объединяются структуры данных и функции их обработки.

Класс содержит константы и переменные, называемые полями, а также операции и функции выполняемые над данными – методы. Доступ к полям класса возможен только через вызов соответствующих методов.

Классы и объекты

Объявление класса:

```
class <имя_класса> {
[<спецификатор_доступа>:]
 <данные и функции>
……..……..
[<спецификатор_доступа>:]
  <данные и функции>
} [<список_объектов>];
```

спецификатор_доступа – это одно из 3-х ключевых слов:

- public – открытая секция класса, открывает доступ к функциям и данным этого класса из других частей программы, это интерфейс класса;
- private – закрытая секция класса, объявляет функции и данные закрытые от внешнего доступа и открытые только членам этого класса, действует по умолчанию;
- protected – защищённая секция класса, этот спецификатор используется только при наследовании классов.
- список_объектов – объявляет объекты класса, можно опускать и объявлять объекты позже.

- Методы, расположенные в открытой части (public), формируют интерфейс класса и могут вызываться другим (внешним) объектом.
- Доступ к закрытой секции класса (private) возможен только из собственных методов класса.
- Доступ к защищённой секции класса (protected) -- из собственных методов класса и из методов классов-потомков данного класса (см. наследование).

Количество одинаковых спецификаторов_доступа не ограничено, порядок их следования – произвольный.

Классы

Пример: объявление (или спецификация) класса:

```
// Файл Book.h – спецификация класса CBook
#pragma once  /*Чтобы компилятор включал объявление типа только один раз при трансляции программы */
class CBook
{
private:     //  спецификатор доступа private действует по умолчанию,
            //  его  можно опускать
   char m_author [50];          // автор
   char *m_pTitle;              // указатель на название
   int m_year;                  // год издания
public:
// методы установки значений
    void setAuthor  const char*);
    void setTitle (const char*);
    void setYear (const int);
// методы возврата значений
      char* getAuthor  void ;
      char* getTitle (void);
      int getYear (void);
}
```

Классы

Данные и функции класса.

Данные класса называются: **поля** (данные-члены, компонентные данные).

Функции класса называются: **методы** (функции-члены, компонентные функции).

Поля и методы – это элементы класса.

Поля класса:

- могут иметь любой тип, кроме типа этого же класса (но могут быть указателями или ссылками на этот класс);
- могут быть описаны с модификатором const, при этом они инициализируются только один раз (с помощью конструктора) и не могут изменяться;
- могут быть описаны с модификатором static, но не как auto, extern и register.

Инициализация полей при описании не допускается.

Классы могут быть ***глобальными*** (объявленными вне любого блока) и ***локальными*** (объявленными внутри блока, например, функции или другого класса).

В каждом классе есть хотя бы один метод, имя которого совпадает с именем класса. Он называется *конструктором* и вызывается автоматически при создании объекта класса. Конструктор предназначен для инициализации объекта. Автоматический вызов конструктора позволяет избежать ошибок, связанных с использованием неинициализированных переменных.

Классы и объекты

Конкретные переменные типа класс называются экземплярами класса, или объектами. Время жизни и видимость объектов зависит от вида и места их описания и подчиняется общим правилам C++:

При создании каждого объекта выделяется память, достаточная для хранения всех его полей, и автоматически вызывается конструктор, выполняющий их инициализацию. Методы класса не тиражируются. При выходе объекта из области действия он уничтожается, при этом автоматически вызывается деструктор.

Доступ к элементам объекта **аналогичен доступу к полям структуры. Для этого используются операция . (точка) при обращении к элементу через имя объекта и операция -> при обращении через указатель, например:**

```
int n = Vasia.get_ammo(); stado[5].draw;
cout << beavis->get_health();
```

Обратиться таким образом можно только к элементам со спецификаторам public. Получить или изменить значения элементов со спецификатором private можно только через обращение к соответствующим методам.

Каждый объект содержит свой экземпляр полей класса. Методы класса находятся в памяти в единственном экземпляре и используются всеми объектами совместно, поэтому необходимо обеспечить работу методов с полями именно того объекта, для которого они были вызваны. Это обеспечивается передачей в функцию скрытого параметра this, в котором хранится константный указатель на вызвавший функцию объект. Указатель this неявно используется внутри метода для ссылок на элементы объекта. В явном виде этот указатель применяется в основном для возвращения из метода указателя (return this;) или ссылки (return *this;) на вызвавший объект.

Указатель this можно также применять для идентификации поля класса в том случае, когда его имя совпадает с именем формального параметра метода.

Классы и объекты

Примеры

```
#include <iostream>
#include <clocale>
using namespace std;

class Number
{  private:  int number;
public:
void setNumber() {
   cout << "Введите число: ";
   cin >> number;  }

void getNumber()
{ cout << number << endl;  }
};

int main()
{  setlocale(LC_ALL, "rus");
     Number object;
    object.setNumber();
   object.getNumber();
return 0;
}
```

Часто определяют члены-методы вне описания класса

```
#include <iostream>
#include <clocale>
using namespace std;
class Number
{
   private:   int number;
public:
  void setNumber();   //прототипы функций
  void getNumber();
};

void Number::setNumber()
{  cout << "Введите число: ";  cin >> number; }

void Number::getNumber()
{   cout << number << endl;  }

int main()
{
  setlocale(LC_ALL, "rus");
      Number object;
    object.setNumber();
    object.getNumber();
 return 0;
}
```

Классы и объекты

Объявление объектов класса

Описание объекта задает его тип (имя класса) и, возможно, необходимые для инициализации членов-данных значения. При объявлении объекта компилятор получает указание на создание переменной класса на основании заданного типа данных (класса).

Когда объект объявляется, то согласно описанию класса для объекта происходит выделение оперативной памяти, а также при указании значений данных осуществляется инициализация членов-данных указанными значениями. Всю эту работу делает специальный метод класса, называемый конструктором.

Основные форматы объявления объекта:

<имя_класса> <имя_объекта>;
<имя_класса> <имя_объекта> (список параметров);
<имя_класса> <имя_объект > (имя_объекта_копирования);

Пример объявления объектов класса:

- **CBook book, aBook [100];**
- **CBook obj ("Carrol L.", "Alice in Wonderland", 2000);**
- **CBook copy (obj);**

Здесь использованы все три формата для объявления объектов типа класса CBook Согласно первому, объявляется объект book и массив из 100 объектов aBook. По второму формату объявлен объект obj, и по третьему -- объект copy.

Классы и объекты

Выбор имён классов

Для облегчения чтения исходного кода рекомендуется придерживаться следующих соглашений по выбору идентификаторов классов:

- **Чтобы не путать имена классов и имена объектов рекомендуется имя класса начинать с заглавной буквы C (Class).**

  Например, CLocomotive, CArray, CStudent.

  [Эти соглашения используются в реализациях C++ фирмы Microsoft, в системах фирмы Borland в качестве префикса имён классов обычно используется заглавная буква T.

  Например, TLocomotive,TArray, TStudent]
- **Файлы спецификации и реализации класса называть именем класса, но без первой буквы C.**
- **Имя члена-данного начинать с префикса m_.**

  Например, m_name, m_state.
- **Метод называть адекватно его назначению.**

  Например, find ( ), isstate ( ).
- **Формальный параметр, имеющий отношение к члену-данному класса, называть точно так же, как и данное, но без префикса m_.**

  Например, name, state.
- **Идентификатор переменной подбирать так, чтобы был ясен физический смысл.**

  Например, если переменная обозначает город, то лучше дать ей имя town, а не dog или xi.

Классы

Указатель this

Указатель, обозначаемый ключевым словом **this**, представляет собой неявно определенный указатель на сам объект и является скрытой внутренней переменной класса. Отсюда следует:

- **this** – это адрес активного объекта в оперативной памяти,
- ***this** – сам активный объект,
- **this -> <имя члена-данного класса>** – указывает на данное активного объекта. Как правило, this -> не пишется для активного объекта в методах класса, которые обрабатывают это данное активного объекта.
- **this -> <имя метода (список фактических параметров)>** – вызывает метод для активного объекта. Как правило, this -> не пишется при вызове метода активным объектом класса,
- получить значение указателя в методах класса можно с помощью ключевого слова **this**,
- **this** – это локальная переменная, недоступная за пределами класса. Узнать значение указателя this какого-либо объекта класса в функциях программы можно только с помощью открытого метода класса, который возвращает это значение, или же просто воспользоваться оператором взятия адреса &.

Абсолютно каждый объект имеет свой собственный указатель **this**. Этот указатель не нужно объявлять в классе, так как объявление

<имя_класса *const this>

скрыто в классе как объявление указателя на константу.

Для любого принадлежащего классу метода this неявно объявлен точно так же.

Изменить this невозможно.

Классы

Оператор разрешения области видимости

Number :: init () /* :: - оператор разрешения области видимости (доступ к области видимости). Он определяет компилятору, что данная версия функции init() принадлежит классу Number, т.е. находится в области видимости этого класса */

Оператор "::" связывает между собой имена класса и его члена, сообщая компилятору, какому классу принадлежит данный член.

Однако оператор разрешения области видимости имеет еще одно предназначение: он открывает доступ к переменной, имя которой "замаскировано" внутри вложенной области видимости.

В качестве примера рассмотрим следующий фрагмент программы:

```
int i;        // Глобальная переменная  i
void f()
 {
int i;        // Локальная переменная  i
i = 10;       // Используем локальную переменную  i
 }
```

Как указано в комментарии, присваивание i = 10 относится к локальной переменной i. А что делать, если функции f() нужен доступ к глобальной переменной i? Тогда следует применить оператор разрешения области видимости.

```
int i;            // Глобальная переменная  i
void f() {
int i;             //Локальная переменная  i
::i = 10; }     // Обращение к глобальной переменной i
```

Классы

Вложенные классы

Один класс можно определить внутри другого – это вложенные классы. Поскольку объявление класса фактически определяет его область видимости, вложенные классы могут существовать только внутри области видимости внешнего класса. Вложенные используются редко, т.к. в языке C++ существует гибкий и мощный механизм наследования и создавать вложенные классы практически не требуется.

Локальные классы

Класс можно определить внутри функции. Пример:

```
#include <iostream>
using namespace std;
void f ();
int main() {
  f ();
// Класс myclass отсюда не виден
  return 0; }
     void f ()
     {  class myclass
        {
           int i;
            public:
              void put_i(int n) { i=n; }
               int get_i()   { return i; }
        } ob;
     ob.put_i(10);
     cout << b.get_i();  }
```

На локальные классы налагается несколько ограничений. Во-первых, все функции-члены должны определяться внутри объявления локального класса. Во-вторых, локальные классы не могут обращаться к локальным переменным, объявленным внутри функции, за исключением статических локальных переменных (static) и внешних переменных (extern). Однако они имеют доступ к именам типов и перечислениям, определенным во внешней функции. В-третьих, внутри локальных классов нельзя объявлять статические переменные.

Классы

Дружественные функции

С помощью ключевого слова **friend** можно предоставить обычной функции доступ к закрытым членам класса. Дружественная функция имеет доступ ко всем закрытым и защищенным членам класса. Для того чтобы объявить дружественную функцию, следует поместить ее прототип внутри класса, указав перед ней ключевое слово friend.

Пример:

```
#include <iostream>
using namespace std;
class myclass  {
     int a, b;
  public:
     friend int sum(myclass x);
     void set_ab(int i,  int j);   };
     void myclass::set_ab(int i,   int j) {a = i; b = j;}
// Внимание: функция sum()  не является функцией-членом никакого класса
 int sum(myclass x) {
/* Так как функция sum()  является дружественной по отношению к классу
myclass,  она имеет прямой доступ к переменным a и b */
return x.a + x.b;       }
int main() {
myclass n; n.set_ab(3, 4); cout << sum(n); return 0;  }
```

В данном примере функция sum() не является членом класса myclass. Однако она обладает полным доступом к закрытым членам. Кроме того, функцию sum() можно вызывать без помощи оператора Поскольку она не является членом класса, ей не нужно указывать имя объекта.

Классы

Дружественные классы

Один класс может быть дружественным по отношению к другому. В этом случае дружественный класс и все его функции-члены имеют доступ к закрытым членам, определенным в другом классе.

Пример:

```
// Применение дружественных классов
#include <iostream>
using namespace std;
  class TwoValues      {
   int a;
   int b;
       public:
    TwoValues(int i,   int j)   {a=i;b=j; }
    friend class Min;   };
 class Min {
      public:
int min(TwoValues x) ;    };
int Min::min(TwoValues x)
  { return x.a < x.b ? x.a : x.b; }

int main() {
  TwoValues ob(10, 20);    Min m;
  cout << m.min(ob);      return 0;   }
```

В данном примере класс Min имеет доступ к закрытым переменным a и b, объявленным в классе TwoValues.

Следует запомнить: если один класс является дружественным по отношению к другому, он просто получает доступ к сущностям, определенным в этом классе, но не наследует их, т.е. члены класса не становятся членами дружественного класса. Дружественные классы редко используются в практических приложениях. Они необходимы лишь в особых случаях.