Классы и объекты

Объектно-ориентированное программирование

- Инкапсуляция и сокрытие данных
- Наследование
- Полиморфизм

Основа ООП - абстрагирование, создание типов, определенных пользователем

Следствие введения классов – расширение возможностей повторного использования кода

Основные термины

- Абстракция
  - Отделяет назначение типа (класса) от его реализации. В основу абстракции положено определение операций над объектами данного типа
- Инкапсуляция
  - Объединяют в единое целое данные (поля) и операции над ними (методы)
- Сокрытие информации
  - Ограничивает возможности работы с полями и методами. Защищает классы от ошибок. Упрощает работу с классами, позволяет усовершенствовать организацию классов

Основные термины

- Наследование
  - Позволяет создавать новые классы путем добавление новых полей и методов к существующим классам, а также изменения методов существующих классов
- Полиморфизм
  - Позволяет выбирать одну из перегруженных операций, в зависимости от объектов к которым она применяется.

Описание класса

```
class ИмяКласса
{
    private:
    //Закрытые элементы класса
    // поля и вспомогательные методы
    public:
    //Открытые элементы
    //прототипы функций или inline реализация
};
```

файл.h

Реализация функций (методов) класса, не определенных как inline

```
тип  ИмяКласса :: имяМетода ( спецификация параметров)
{
  //Тело метода
}
```

файл.cpp

Использование класса

- Класс – пользовательский тип данных
- Переменные типа называются объектами
- C++ две модели объектов:
  - простая
  - ссылочная
- Объявление

ИмяКласса имяОбъекта;

*ИмяКласса * имяУказателяНаОбъект;*

Пример

```
class Timer {
  private:
    int hour, minute, second;
  public:
    void  setTime (int h, int m, int s);
    void  tic ();
    int     getHour();
    void  setHour (int h);
    void  nulled();
     int    timeInSec();
     int    period ( Timer x);
. . .
};
```

Пример

```
int   Timer :: period ( Timer x)
{
   return timeInSec( ) – x.timeInSec();
. . .
}
```

Использование

```
Timer a, b;
 a.setHour(15); …
 b.nulled();
 cout << a.period(b);
```

Особые методы

- Конструкторы
  - не имеет возвращаемого значения
  - имя конструктора совпадает с именем класса
  - класс может иметь несколько перегруженных конструкторов
  - класс всегда имеет 2 конструктора – без параметров и конструктор-копии
- Деструкторы
  - не имеет возвращаемого значения и параметров
  - имя начинается с ~ затем следует имя класса
  - всегда один и всегда есть

Пример

```
class Timer {
. . .
public:
    Timer () { hour= minute = second =0;}
    Timer ( int h, int m=0, int s=0 ) :
            hour(h), minute(m), second(s) { }
    Timer (const Timer & t);
    ~Timer (){}
};
```

Когда выполняются конструкторы

- Без параметров

```
Timer a;
Timer b = Timer ();
Timer *c = Timer Point;
Timer  d[10];
Timer *f = new Timer[10];
```

Когда выполняются конструкторы

- С параметрами

```
Timer  lect(14,10, 0);
Timer  b= Timer (14, 10,0);
Timer  *c=new Timer(13,30, 30);
Timer  audit[2] ={Timer(10,15,0), Timer(12, 20,0)};

Timer  a(13);
Timer  b= Timer(12, 20);
Timer  c=15;
```

- ***специальная форма записи конструктора с параметрами – список инициализации***

```
Timer ( int h, int m=0, int s=0 ) :
            hour(h), minute(m), second(s) { }
```

Когда выполняются конструкторы

- конструктор копии всегда присутствует
- если он создается по умолчанию, то выполняет поверхностное копирование (поэлементное копирование). Поверхностное копирование создает копии простых типов и указателей
- для полного копирования (глубокого), например, массивов, строк, списков, необходимо описывать собственный конструктор копии

Когда выполняются конструкторы

- конструктор копии

// Timer b;

Timer a(b);
Timer a=b;
Timer a= Timer(b);
Timer *a = new Timer(b);

Конструктор копирования вызывается при передаче параметров в функции по значению и при возврате значения из функции

Timer myFunc (Timer x);

Когда выполняются деструкторы

- при освобождении памяти, занимаемой объектом

```
{
    Timer x;
    Timer *y = new Timer;

     delete y;   // деструктор для y

} // деструктор для x
```

Когда выполняются деструкторы

- при освобождении памяти, занимаемой объектом

```
Timer  myFunc ( Timer x) // конструктор копии
{
    Timer y (x);  // конструктор копии
….
     return y; // конструктор копии
} // деструктор для y  и x
```

Указатель this

- Каждый метод класса получает один неявный параметр – указатель на тот объект, который вызвал этот метод. Этот указатель имеет имя this

```
void  Timer ::nulled() {
   hour = 0;
   minute = 0;
   second = 0;
}
```

```
void  Timer ::nulled() {
   this->hour = 0;
   this->minute = 0;
   this->second = 0;
}
```

Указатель this

```
int   Timer :: period (Timer    x)
{
  return timeInSec( ) – x.timeInSec();
}
```

```
int   Timer :: period (Timer    x)
{
  return  this -> timeInSec( ) – x.timeInSec();
}
```

Указатель this

```
Timer   Timer :: maxTime ( const Timer&    x)
{
   if (period(x) < 0) return  x;
        else return ????;
}
```

```
Timer   Timer :: maxTime ( const Timer&    x)
{
   if (period(x) < 0) return  x;
       else return   *this;
}
```

Функции для доступа к закрытым полям класса

- Традиционно имеют имена setXXX и getXXX, где XXX – имя соответствующего поля.
- Отсюда их названия «сеттеры» и «геттеры».