Технологии программирования

Доц. каф. «Медиаменеджмента и медиапроизводства» Евич Л.Н.

Лекция 14. ООП в C++.

Инкапсуляция.

Инкапсуляция (encapsulation) - это механизм, который объединяет данные и код, манипулирующий этими данными, а также защищает и то, и другое от внешнего вмешательства или неправильного использования.

Сокрытие информации – это формальный механизм, предотвращающий прямой доступ к внутреннему представлению типа класса из функций программы.

Ограничение доступа к членам задается с помощью секций тела класса, помеченных ключевыми словами public, private и protected – спецификаторами доступа.

Члены, объявленные в секции public, называются открытыми, а объявленные в секциях private и protected соответственно закрытыми или защищенными.
открытый член доступен из любого места программы.
понятие наследования.)

Лекция 14. Классы в C++.

```
//MyClass1.cpp
#include <iostream.h >

class MyClass {
int a;
public:
   MyClass() { a=5;}    //конструктор
   void Show() { cout<< a; }
};

int main() {
      MyClass ob;
      ob.a=10;
      ob.Show();
      return 0;
}
```

Лекция 14. Классы в C++.

```
#include <iostream.h >

class MyClass {
public:
   MyClass() { a=10;}    //конструктор
   void Show() { cout<< a; }
private:
   int a;
};

int main() {
      MyClass ob;  int k;
      ob.a=10;              // Недопустимо
      k=ob.a+3;       // Недопустимо
      cout << ob.a;       // Недопустимо
      return 0;
}
```

Лекция 14. Классы в C++.

// Все действия с закрытыми свойствами класса реализуются через его методы.

```
#include <iostream.h >
 class MyClass {
int a;
public:
   MyClass() { a=10;}    //конструктор
   void Show() { cout<< a; }
};

int main() {
      MyClass ob;
      ob.a=10;              // Недопустимо
      ob.Show();
      return 0;
}
```

Лекция 14. Классы в C++.

```
//MyClass1.cpp
#include <iostream.h >

class MyClass {
private:
   int a;
public:
   MyClass() { a=10;}    //конструктор
   void Show() { cout<< a; }
   void get(int x) {a=x; }
};
int main() {
      MyClass ob;  int k;
      cin >> k;
      ob.get(k);
      ob. Show() ;
      return 0;
}
```

Лекция 14. Классы в C++.

Чтобы не было "покушений" на константность объекта, компилятор должен различать безопасные (те, которые не изменяют объект) и небезопасные (те, которые пытаются это сделать) функции-члены.
Проектировщик класса может указать, какие функции-члены не модифицируют объект, объявив их константными с помощью спецификатора const:

```
#include <iostream.h>
 class MyClass {
private:
      int a;
 public:
   MyClass();    //конструктор
   ~MyClass();   //деструктор
   void Show() const;
};

void MyClass::Show() const  {
      cout << a << endl;
}
```

Лекция 14. ООП в C++.

Инкапсуляция.

Обычно, приватными делают все свойства класса, а публичными — его методы.
Класс, скрывающий информацию, оставляет открытыми только функции-члены, определяющие операции, с помощью которых внешняя программа может манипулировать его объектами;
закрытый член доступен только функциям-членам и друзьям класса.

Класс, который хочет скрыть информацию, объявляет свои данные-члены закрытыми;
защищенный член ведет себя как открытый по отношению к производному классу и как закрытый по отношению к остальной части программы.

Лекция 14. ООП в C++.

Наследование.

Наследование позволяет создавать производные классы (классы наследники), взяв за основу все методы и элементы базового класса (класса родителя).

Объекты производного класса могут свободно использовать всё, что создано и отлажено в базовом классе. При этом, мы можем в производный класс, дописать необходимый код для усовершенствования программы: добавить новые элементы, методы и т.д.. Базовый класс останется нетронутым.

Лекция 14. ООП в C++.

Наследование.

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

class FirstClass    // базовый класс
{
protected:          // спецификатор доступа к элементу a
    int a;
public:
    FirstClass()  {
        a= 0;
    }
     FirstClass( int x)  {
        a = x;
    }
     void show ()  {
        cout << a << endl;
    }
};
```

Типы наследования и типы доступа

Лекция 14. ООП в C++. Наследование.

```
class FirstClass  {   // базовый класс
protected:          // спецификатор доступа к элементу a
    int a;
public:
    FirstClass()  { a= 0; }
     void show ()  { cout << a << endl; }
};
class SecondClass : public FirstClass  {  // производный класс
…
};
```

Способ доступа	Спецификатор в базовом классе	Доступ в производном классе
private	private protected public	нет private private
protected	private protected public	нет protected protected
public	private protected public	нет protected public

Лекция 14. ООП в C++. **Наследование.**

```
#include <iostream>
using namespace std;
class FirstClass  {   // базовый класс
protected:          // спецификатор доступа к элементу a
    int a;
public:
    FirstClass()  { a= 0; }
    FirstClass( int x)  { a = x; }
     void show ()  { cout << a << endl; }
};
class SecondClass : public FirstClass  {  // производный класс
public:
    SecondClass() : FirstClass ()    // конструктор класса SecondClass вызывает конструктор класса FirstClass
    { }
     SecondClass(int b) : FirstClass (b)    // inputS передается в конструктор с параметром класса FirstClass
    { }
     void ValueSqr () // возводит a  в квадрат.
                                //   Без спецификатора доступа protected эта функция не могла бы изменить значение a
    {  a*= a; }
};
```

Лекция 14. ООП в C++. Наследование.

```
#include <iostream>
using namespace std;
class FirstClass  {   // базовый класс
protected:          // спецификатор доступа к элементу a
    int a;
public:
    FirstClass()  { a= 0; }
    FirstClass( int x)  { a = x; }
     void show ()  { cout << a << endl; }
};
class SecondClass : public FirstClass  {  // производный класс
public:
    SecondClass() : FirstClass ()  { } // конструктор класса SecondClass вызывает конструктор класса FirstClass
    SecondClass(int b) : FirstClass (b) { } // inputS передается в конструктор с параметром класса FirstClass
        …
};
```

Конструкторы не наследуются, поэтому производный класс должен иметь собственные конструкторы.

Производный класс не может обратиться к конструктору с параметрами. Если конструкторы в производном классе не определены, при создании объекта сработает конструктор без аргументов базового класса. А если нам надо сразу при создании объекта производного класса внести данные, то для него необходимо определить свои конструкторы.

Лекция 14. ООП в C++. Наследование.

```
#include <iostream>
using namespace std;
class FirstClass  {   // базовый класс
protected:          // спецификатор доступа к элементу a
    int a;
public:
    FirstClass()  { a= 0; }
    FirstClass( int x)  { a = x; }
     void show ()  { cout << a << endl; }
};
class SecondClass : public FirstClass  {  // производный класс
public:
    SecondClass() : FirstClass ()    { }
     SecondClass(int b) : FirstClass (b)  { }
     void ValueSqr ()  {  a*= a; }
};
void main() {
    FirstClass Object1(3);     // объект базового класса
    cout << “Object1 = ";
    Object1.show ();
    SecondClass Object2(4);    // объект производного класса
    cout << "value Object2 = ";
    Object2.show ();  // вызов метода базового класса
    Object2.ValueSqr();        // возводим a в квадрат
    cout << "квадрат value Object2 = ";
    Object2.show ();
     //Object1.ValueSqr();       // базовый класс не имеет доступа к методам производного класса
    cout << endl;
}
```

Лекция 14. ООП в C++. Наследование.

- ✔ Наследование — это определение производного класса, который может обращаться ко всем элементам и методам базового класса за исключением тех, которые находятся в поле private;
- ✔ Производный класс еще называют потомком или подклассом, а базовый — родитель или надкласс; Синтаксис определения производного класса:

class Имя_Производного_Класса : специф.доступа Имя_Базового_Класса
{ /*код*/ } ;

- ✔ Производный класс имеет доступ ко всем элементам и методам базового класса, а базовый класс может использовать только свои собственные элементы и методы.
- ✔ В производном классе необходимо явно определять свои конструкторы, деструкторы и перегруженные операторы присваивания из-за того, что они не наследуются от базового класса. Но их можно вызвать явным образом при определении конструктора, деструктора или перегрузки оператора присваивания производного класса, например таким образом (для конструктора):

Конструктор_Производного_Класса
(/*параметры*/) : Конструктор_Базового_Класса (/*параметры*/) { } .

Лекция 14. ООП в C++.

Полиморфизм.

Полиморфизм (polymorphism) (от греческого polymorphos) - это свойство, которое позволяет одно и то же имя использовать для решения двух или более схожих, но технически разных задач. Целью полиморфизма, применительно к объектно-ориентированному программированию, является использование одного имени для задания общих для класса действий.

Полиморфизм, характеризуется следующей фразой: «один интерфейс — множество методов».

Лекция 14. ООП в C++.

Полиморфизм.

Для изменения метода необходимо перегрузить его в потомке, т.е. объявить в потомке одноименный метод и реализовать в нем нужные действия.

В результате в объекте-родителе и объекте-потомке будут действовать два одноименных метода, имеющие разную кодовую реализацию и, следовательно, придающие объектам разное поведение.

Лекция 14. ООП в C++. Наследование.

Перегруженные методы класса потомка

```
#include <iostream>
using namespace std;
class FirstClass  {   // базовый класс
protected:
    int a;
public:
    FirstClass()  { a= 0; }
    FirstClass( int x)  { a = x; }
    virtual void show ()  { cout << a << endl; }
};
```

```
void main() {
    FirstClass Object1(3);
    cout << “Object1 = ";
    Object1.show ();
    SecondClass Object2(4);
    cout << "value Object2 = ";
    Object2.show ();  // вызов метода
                      производного класса
}
```

```
class SecondClass : public FirstClass  {  // производный класс
public:
    SecondClass() : FirstClass ()  { }
    SecondClass(int b) : FirstClass (b)  { }
    virtual void show ()  { cout << “Функция производного класса”<< a << endl; }
};
```

Лекция 14. Классы в C++.

Задание 2

1. Добавить в класс четыре метода класса (get, set, любые)
2. Две внешние функции;
3. Проиллюстрировать в программе использование всех методов:
    1) Передача данных из программы в поля класса
    2) Из полей класса в программу
    3) С помощью внешнего метода обратиться к методам класса
    4) Организовать массив объектов. Передать данные в массив
4. Создать наследника с двумя конструкторами и двумя методами, один из которых переопределен от базового класса.
5. Проиллюстрировать работу наследника.