Объектно-ориентированное программирование

Курс «Разработка ПО систем управления»

Кафедра управления и интеллектуальных технологий НИУ «МЭИ»

Весна 2021 г.

ООП

- Объектно-ориентированное программирование (ООП) - это популярная парадигма программирования, пришедшая на смену процедурному подходу в программировании.
- Процедурное программирование - это монолитная программа с набором инструкций для выполнения, с ветвлениями и подпрограммами

Понятия ООП

- Класс – модель категории вещей со свойствами (полями, данными-членами) и действиями над ними (методами, функциями-членами).
- Объект – представитель класса, одна конкретная вещь из категории.
- Класс – тип данных, объект – значение этого типа.

Класс	Человек	Стол	vector <int>
Объект 1:	Иванов И.И.	Первая парта	xs
Объект 2:	Петров В.В.	Последняя парта	ys
Объект 3:	Сидоров Н.Н.	Стол у окна	{1, 2, 3}

Классы, структуры, объекты

- Структура = Класс (с некоторыми различиями)

- Технически все просто:
  - Класс = данные (поля) + функции (методы)
  - Класс похож на структуру и набор функций над ней

- Правила «хорошего кода»:
  - Структура – для описания объекта (данные)
  - Класс – если совершаем какие-то действия (методы)

Пример класса

```
class Student
{
    public:
        string name;
        int year;
        void study () {
                    cout << “Student “ <<  name << “ is studying” << endl;
                    }
};
int main()
{
            Student student;
            student.name = “Tom”;
            student.year = 2017;
            student.study();    // Student Tom is studying
}
```

Инкапсуляция

- Объект управляется своим состоянием, оно скрыто и не может быть испорчено из внешнего кода.
- В C++ реализуется через уровни доступа
  - **public** – члены доступны извне,
  - **private** – только из класса
  - **protected** - из класса и из наследуемого класса (рассмотрим далее)

Разница между struct и class – в уровне доступа по умолчанию:

```
struct X {
//…

}
```

```
class X {
        public:
        //…
}
```

```
struct X {
private:
//…
}
```

```
class X {

        //…
}
```

Инкапсуляция (2)

```
class Student
{
        string name;
        int year;
public:
        void study () {
                cout << “Student “ <<  name << “ is studying” << endl;
                }

        void set_name (string n) {
             // проверка введенного значения и только после – изменение поля name
            name = n;
          }
};
int main()
{
             Student student;
             student.set_name(“Tom”);   //student.name = “Tom”;  - Ошибка!
             student.study();   // Student Tom is studying
}
```

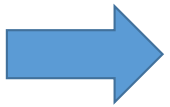
Наследование

- Наследование - процесс, посредством которого один объект может приобретать свойства другого.
  - объект может наследовать основные свойства другого объекта (base class, класс-родитель) и добавлять к ним черты, характерные только для него (derived class, класс-потомок)

  - Модификатор доступа **protected** – позволяет получать доступ к членам класса в классе-потомке

Наследование

```
class Person
{
public:
   string name;      //  имя
   int age;          // возраст
   void display()
   {
      cout << "Name: " << name << "\tAge: " << age << endl;
   }
};
class Student
{
public:
   string name;      // имя
   int age;          // возраст
   string group;     // учебная группа
   void display()
   {
      cout << "Name: " << name << "\tAge: " << age << endl;
   }
};
```



```
class Person
{
public:
   string name;      //  имя
   int age;          // возраст
   void display()
   {
      cout << "Name: " << name << "\tAge: " << age << endl;
   }
};
class Student: public Person
{
public:
   string group;    // группа
};
```

Полиморфизм

- Способность, позволяющая использовать одно и тоже имя функции для решения двух и более схожих, но технически разных задач.

- Возможность замещения методов объекта родителя методами объекта-потомка, имеющих то же имя. Другими словами, определение того, какую версию метода текущего объекта следует запустить.
  Например, у вас есть несколько наследуемых классов, с разной реализацией одного и того же метода. То есть, у вас есть несколько разных методов с одинаковым именем (и сигнатурой), реализованных в разных классах. Какую версию метода вызвать? Это зависит от типа переменной, в которой находится объект.

Полиморфизм

```
class Animal {
    public:
        void Info() { cout << "Animal"; }
}

class Dog : Animal {
    public:
        void Info() { cout << "Dog"; }
}
```

```
Animal dog1;
Dog dog2;

// вызовется метод класса, указанного у переменной
dog1.Info(); // напишет Animal
dog2.Info(); // напишет Dog
```