Программирование на C++

Минцифры России

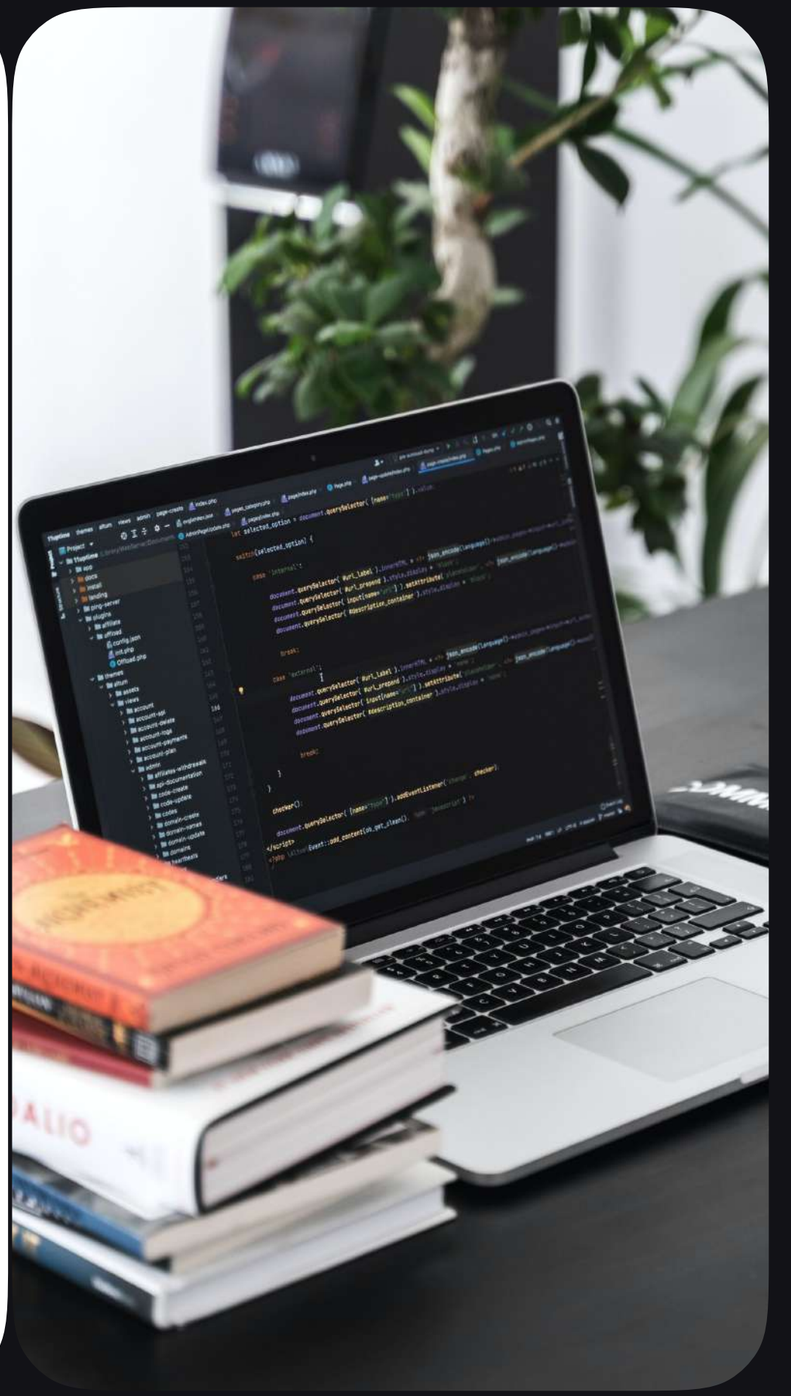
UCHi DOMA

20.35 УНИВЕРСИТЕТ

Урок 6 Модуль 3

Отношения дружественности и наследования между классами

Полезные материалы



Цели урока

- изучить отношения дружественности и наследования между классами
- отработать на практике написание алгоритмов с дружественными классами, функциями и наследованием на C++

Дружественные функции

Некоторые функции нуждаются в привилегированном доступе более, чем к одному классу. В этом случае требуется, чтобы функция — не член класса, имела доступ к скрытым членам класса.

Такие функции получили название **дружественных.**

Дружественные функции

Для того, чтобы функция — не член класса имела доступ к **private**-членам класса, необходимо в определение класса поместить объявление этой дружественной функции, используя ключевое слово friend.

```
void func() {...}
class A
{
    ...
    friend void func();
};
```

Объявление дружественной функции начинается с ключевого слова **friend** и должно находиться только в определении класса.

Дружественные функции

Дружественная функция, хотя и объявляется внутри класса, методом класса не является. Поэтому не имеет значения, в какой части тела класса (**private**, **public**) она объявлена.

Дружественные функции

Метод одного класса может быть дружественным для другого класса.

```
class A
{
    ...
    int func();
};
class B
{
    ...
    friend int A :: func();
};
```

Метод **func()** класса A является дружественным для класса **B**.

Дружественный класс

Если все методы одного класса являются дружественными для другого класса, то можно объявить дружественный класс:

```
class A {
    ...
};
class B
{
    ...
    friend class A;
};
```

friend class ИмяКласса;

Все методы класса A будут иметь доступ к скрытым членам класса B.

Пример дружественной функции

```
#include <iostream>
using namespace std;
class Summator
{
private:
    int m_value;
public:
    Summator() { m_value = 0; }
    void add(int value) { m_value += value; }
    int getValue(){return m_value;}

    // Сделаем функцию reset() другом этого класса
    friend void reset(Summator &summator);
};
// reset() теперь является другом класса Summator
void reset(Summator &summator)
{
    // и может получить доступ к закрытым данным объектов Summator
    summator.m_value = 0;
}
int main()
{
    Summator a;
    a.add(5); // добавляем 5 в накапливающий сумматор
    cout<<a.getValue()<<endl;
    reset(a); // сбрасываем накапливающий сумматор в 0
    cout<<a.getValue()<<endl;
    return 0;
}
```

Пример дружественной функции

В этом примере мы объявили функцию с именем **reset()**, которая принимает объект класса **Summator** и устанавливает значение **m_value** равным 0. Поскольку **reset()** не является членом класса **Summator** обычно **reset()** не будет иметь доступ к закрытым членам **Summator**. Однако, поскольку **Summator** специально объявил эту функцию **reset()** как друга класса, то ей предоставляется доступ к закрытым членам **Summator**.

Обратите внимание, что мы должны передать в **reset()** объект **Summator**. Это потому, что **reset()** не является функцией-членом.

Несколько друзей

```
#include <iostream>
using namespace std;
class Humidity;
class Temperature
{
private:
    int m_temp;
public:
    Temperature(int temp=0) { m_temp = temp; }
    friend void printWeather(const Temperature &temperature,
                             const Humidity &humidity);
};
class Humidity
{
private:
    int m_humidity;
public:
    Humidity(int humidity=0) { m_humidity = humidity; }
    friend void printWeather(const Temperature &temperature,
                             const Humidity &humidity);
};
void printWeather(const Temperature &temperature, const Humidity &humidity)
{
    cout << "Температура: "<< temperature.m_temp<<endl;
    cout << "Влажность: " << humidity.m_humidity;
}
int main()
{
    Humidity hum(10);
    Temperature temp(5);
    printWeather(temp, hum);
    return 0;
}
```

Функция может быть другом для более чем одного класса одновременно.

Несколько друзей

- Поскольку **printWeather** является другом обоих классов, она может получить доступ к закрытым данным из объектов обоих классов.
- **class Humidity**; Это прототип класса, который сообщает компилятору, что в будущем мы собираемся определить класс под названием **Humidity**.
- Без этой строки компилятор при синтаксическом анализе прототипа для **printWeather()** внутри класса **Temperature** сообщит, что **Humidity** не определено.
- Прототипы классов выполняют ту же роль, что и прототипы функций — они сообщают компилятору, как что-то выглядит, чтобы его можно было использовать сейчас и определить позже.
- Однако, в отличие от функций, классы не имеют возвращаемых типов или параметров, поэтому прототипы классов всегда представляют собой просто **class ClassName**, где **ClassName** — это имя класса.

Пример дружественного класса

```cpp
#include <iostream>
#include <cstring>
#include <string>
using namespace std;
class Storage
{
private:
    char fname[128];
    char lname[128];
public:
    Storage(char f[],char l[])
    {
        strcpy(fname, f);
        strcpy(lname, l);
    }
    friend class Display;  // Сделаем класс Display другом Storage
};

class Display
{
private:
    bool m_displayIntFirst;
public:
    Display(bool displayIntFirst) { m_displayIntFirst = displayIntFirst;
}
    void displayItem(const Storage &storage)
    {
        if (m_displayIntFirst) // сначала отображаем Фамилию
            cout << storage.fname << ' ' << storage.lname << '\n';
        else // сначала отображаем Имя
            cout << storage.lname << ' ' << storage.fname << '\n';
    }
};

int main()
{
    Storage s("Иван","Петров");
    Display display(true);
    display.displayItem(s);
    return 0;
}
```

Поскольку класс **Display** является другом **Storage**, любой из членов **Display**, использующих объект класса **Storage**, может напрямую обращаться к закрытым членам Storage.

Дружественный класс

- То, что Display является другом Storage, не означает, что Storage также является другом Display.
- Если необходимо, чтобы два класса дружили друг с другом, они оба должны объявить друг друга друзьями.
- Например, если класс A является другом B, а B — другом C, это не означает, что A является другом C.

Наследование

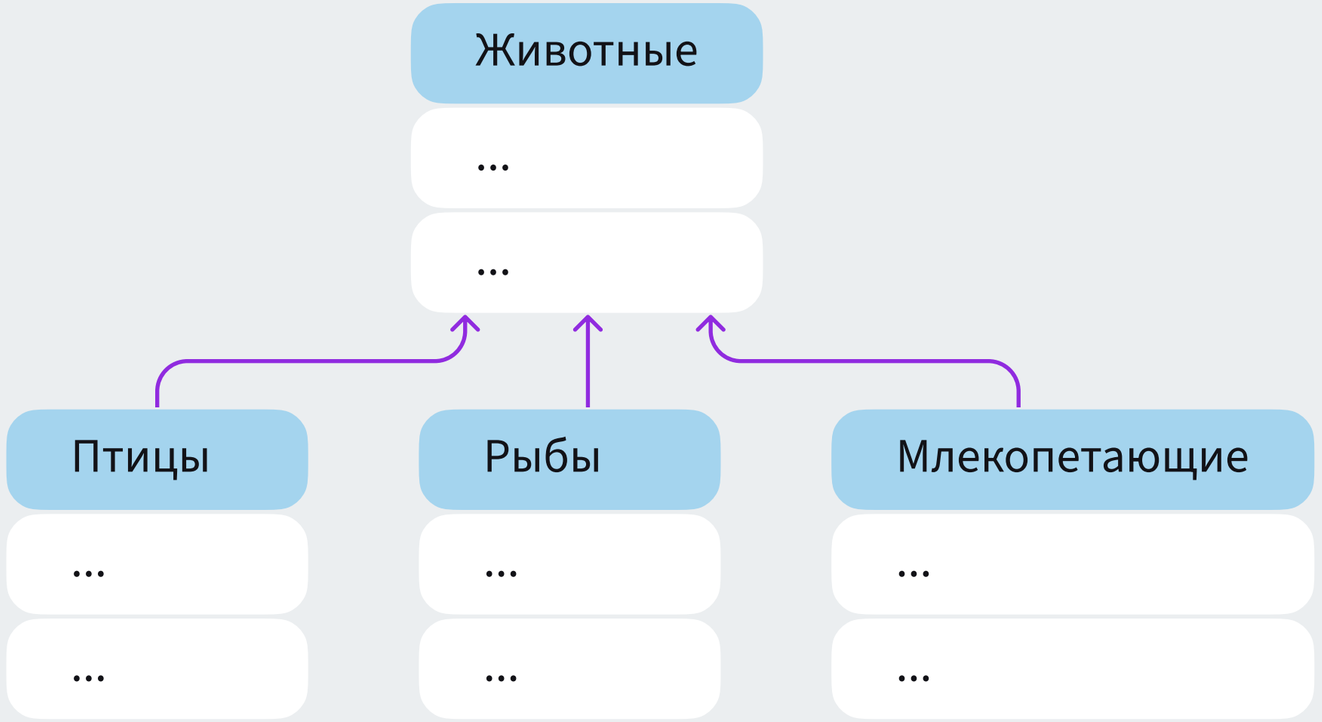
Наследование — это механизм создания нового класса на основе уже существующего. При этом к существующему классу могут быть добавлены новые элементы (данные и функции), либо существующие функции могут быть изменены.

Основное назначение механизма наследования — повторное использование кодов.

Объекты разных классов и сами классы могут находиться в отношении наследования, при котором формируется иерархия объектов, соответствующая заранее предусмотренной иерархии классов.

Наследование

Иерархия классов позволяет определять новые классы на основе уже имеющихся. Имеющиеся классы обычно называют **базовыми**(иногда порождающими), а новые классы, формируемые на основе базовых, — **производными** (порожденными, классами-потомками или наследниками).



Наследование

Производные классы «получают наследство» — данные и методы своих базовых классов, и могут пополняться собственными компонентами (данными и собственными методами).

Для порождения нового класса на основе существующего используется следующая общая форма

```
class Имя : МодификаторДоступа  ИмяБазовогоКласса
{ объявление_членов;};
```

Наследование

При объявлении порождаемого класса **МодификаторДоступа** может принимать значения **public**, **private**, **protected** либо отсутствовать, по умолчанию используется значение **private**.
В любом случае порожденный класс наследует все члены базового класса, но доступ имеет не ко всем. Ему доступны общие (**public**) члены базового класса и недоступны частные (**private**).

Для того, чтобы порожденный класс имел доступ к некоторым скрытым членам базового класса, в базовом классе их необходимо объявить со спецификацией доступа защищенные (**protected**).

Члены класса с доступом **protected** видимы в пределах класса и в любом классе, порожденном из этого класса.

Общее наследование

При общем наследовании порожденный класс имеет доступ к наследуемым членам базового класса с видимостью **public** и **protected**. Члены базового класса с видимостью **private** — недоступны.

Спецификация доступа	Внутри класса	В порождённом классе	Вне класса
private	+	–	–
protected	+	+	–
public	+	+	+

Общее наследование

Общее наследование означает, что порожденный класс — это подтип базового класса. Таким образом, порожденный класс представляет собой модификацию базового класса, которая наследует общие и защищенные члены базового класса.

Пример наследования

```
class User
{
    public:
        int Id;
        char Name[128];
};

class Manager : public User
{
    public: char Company[128];
};
```

Объекты класса Manager также являются и объектами класса User.

Пример наследования

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

class User
{
    public:
        int Id;
        void SetName(char n[]) { strcpy(Name,n);}
    protected:
        char Name[128];
};

class Manager : public User
{
    public: char Company[128];
};
int main()
{
    Manager a;
    a.Id=1;
    a.SetName("Менеджер");
    strcpy(a.Company,"Trade");
}
```

Конструкторы и деструкторы при наследовании

Как базовый, так и производный классы могут иметь конструкторы и деструкторы.

Если и у базового и у производного классов есть конструкторы и деструкторы, то конструкторы выполняются в порядке наследования, а деструкторы — в обратном порядке.
То есть если A — базовый класс, B — производный из A, а C — производный из B (A-B-C), то при создании объекта класса C вызов конструкторов будет иметь следующий порядок:

- конструктор класса A
- конструктор класса B
- конструктор класса C

Вызов деструкторов при удалении этого объекта произойдет в обратном порядке:

- деструктор класса C
- деструктор класса B
- деструктор класса A

Пример конструкторов при наследовании

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

class User
{
    public:
        int Id;
        User(int id,char n[]) { Id=id; strcpy(Name,n);} //Конструктор базового класса
        void printName(){cout<<Name<<endl;}
    protected:
        char Name[128];

};

class Manager : public User
{
    public:
    char Company[128];
    //Конструктор класса потомка
    Manager(int id, char n[],char com[]):User(id,n) {strcpy(Company,com);}
};
int main()
{
    Manager a(1,"Компания1","Иван");
    cout<<a.Id<<endl;
    cout<<a.Company<<endl;
    a.printName();
}
```