C++

Урок 13

Разбор ДЗ

Повтороение

Неявный указатель this

Для чего нужен this?

Наследование

Что такое наследование?

наследование

Наследование — парадигма ООП, при которой дочерний объект получает те же поля и методы, что и в базовом классе.

Наследование позволяет определить базовый класс для определенных функций (доступа к данным или действий), а затем создавать производные классы, которые наследуют или переопределяют функции базового класса.

наследование

Базовый класс – тот, от которого реализуется механизм наследования

Дочерний класс – потомок базового класса.

наследование

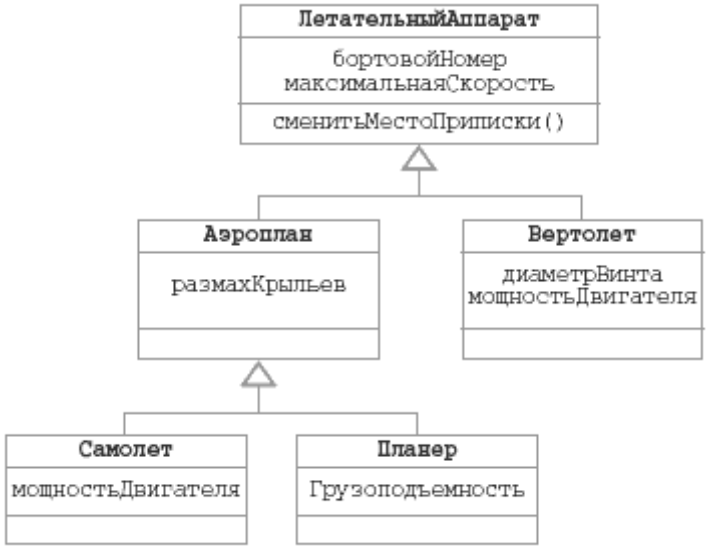
Базовый класс – тот, от которого реализуется механизм наследования

Дочерний класс – потомок базового класса. (получает те же поля и методы, что и в базовом классе)

наследование

Базовый класс – тот, от которого реализуется механизм наследования

Дочерний класс – потомок базового класса. (получает те же поля и методы, что и в базовом классе)



наследование

```
class A{

};

class B: [доступ] A{ //B - дочерний класс от класса A

};
```

Синтаксис наследования

наследование

Модификаторы наследования:

public – публичные члены базового класса доступны. Приватные члены базового класса недоступны. Proteted члены доступны внутри дочернего класса.

private – задается по умолчанию, может отсутствовать. И публичные и приватные члены базового класса недоступны.

protected – в базовом классе элементы, объявленные как protected, снаружи класса трактуются как private. Но в классах-наследниках эти поля доступны.

наследование

Модификатор доступа	Модификатор наследования		
	Private	Protected	Public
Private	Нет доступа	Нет доступа	Нет доступа
Protected	Private	Protected	Protected
Public	Private	Protected	Public

наследование

Protected – модификатор доступа, который позволяет получить доступ к полям и методам базового класса из дочернего.

наследование

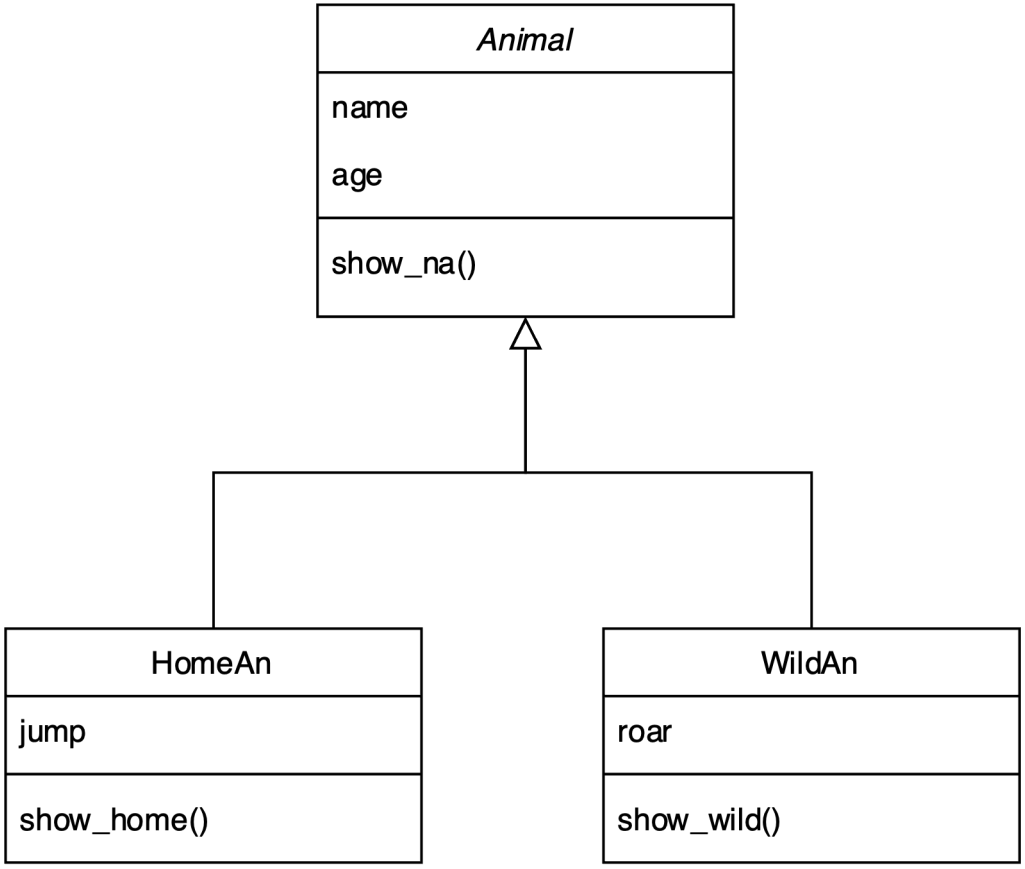


Диаграмма классов

наследование

```
class Animal
{
    int age;
    string name;

public:
    Animal(string name = " ", int age = 0)
    {
        this->name = name;
        this->age = age;
    }

    void show_na()
    {
        cout << name << " " << age << endl;
    }
};
```

Класс Animal

наследование

```
class HomeAn : public Animal
{
    double jump;

public:
    HomeAn(string name = " ", int age = 0, double jump = 0) : name(name), age(age), jump(jump) {}

    void show_home()
    {
        // cout << name << " " << age << " " << jump << endl;
        show_na(); //используем ф-ию базового класса
        cout << " " << jump;
    }
};
```

Класс HomeAn

наследование

```
class HomeAn : public Animal
{
    double jump;

public:
    HomeAn(string name = " ", int age = 0, double jump = 0) : name(name), age(age), jump(jump) {}

    void show_home()
    {
        // cout << name << " " << age << " " << jump << endl;
        show_na(); //используем ф-ию базового класса
        cout << " " << jump;
    }
};
```

```
ge = 0, double jump = 0) : name(name), age(age), jump(jump) {}
```

std::string Animal::name

член "Animal::name" (объявлено в строке 10) недоступно C/C++(265)

Просмотреть проблему Быстрое исправление... (⌘.)

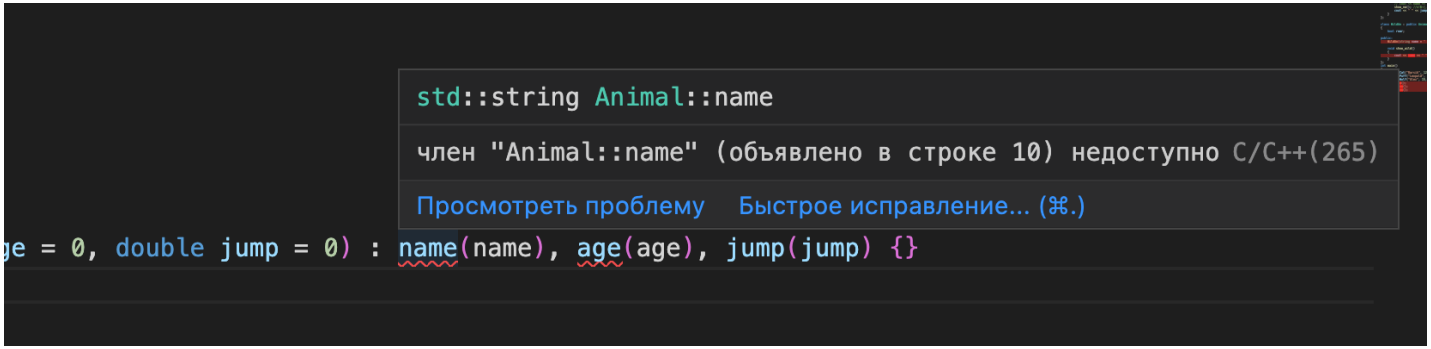
Класс HomeAn

наследование

```
class HomeAn : public Animal
{
    double jump;

public:
    HomeAn(string name = " ", int age = 0, double jump = 0) : name(name), age(age), jump(jump) {}

    void show_home()
    {
        // cout << name << " " << age << " " << jump << endl;
        show_na(); //используем ф-ию базового класса
        cout << " " << jump;
    }
};
```



Как исправить ошибку?

наследование

```
class Animal
{
protected: // модификатор доступа, который дает доступ к полям age, name в дочернем классе
    int age;
    string name;

public:
    Animal(string name = " ", int age = 0)
    {
        this->name = name;
        this->age = age;
    }

    void show_na()
    {
        cout << name << " " << age << endl;
    }
};
```

Класс Animal

наследование

```cpp
class Animal
{
protected: // модификатор доступа, который дает доступ к полям age, name в дочернем классе
    int age;
    string name;

public:
    Animal(string name = " ", int age = 0)
    {
        this->name = name;
        this->age = age;
    }

    void show_na()
    {
        cout << name << " " << age << endl;
    }
};
```

```cpp
class HomeAn : public Animal
{
    double jump;

public:
    HomeAn(string name = " ", int age = 0, double jump = 0) : name(name), age(age), jump(jump) {}

    void show_home()
    {
        show_na(); //используем ф-ию базового класса
        cout << " " << jump;
    }
};
```

std::string Animal::name

"name" не является нестатическим элементом данных или базовым классом для класса "HomeAn" C/C++ (292)

Просмотреть проблему Быстрое исправление... (⇧⌘7)

Класс Animal + HomeAn

наследование

```
class Animal
{
protected: // модификатор доступа, который дает доступ к полям age, name в дочернем классе
    int age;
    string name;

public:
    Animal(string name = " ", int age = 0)
    {
        this->name = name;
        this->age = age;
    }

    void show_na()
    {
        cout << name << " " << age << endl;
    }
};
```

```
class HomeAn : public Animal
{
    double jump;

public:
    HomeAn(string name = " ", int age = 0, double jump = 0) : name(name), age(age), jump(jump) {}

    void show_home()
    {
        show_na(); //используем ф-ию базового класса
        cout << " " << jump;
    }
};
```

```
std::string Animal::name

"name" не является нестатическим элементом данных или базовым классом для класса "HomeAn" C/C++ (292)

Просмотреть проблему  Быстрое исправление... (⇧⌘7)
```

Как решить проблему?

наследование

```
class Animal
{
protected: // модификатор доступа, который дает доступ к полям age, name в дочернем классе
    int age;
    string name;

public:
    Animal(string name = " ", int age = 0)
    {
        this->name = name;
        this->age = age;
    }

    void show_na()
    {
        cout << name << " " << age << endl;
    }
};

class HomeAn : public Animal
{
    double jump;

public:
    HomeAn(string name = " ", int age = 0, double jump = 0) : Animal(name, age), jump(jump) {}

    void show_home()
    {
        show_na(); //используем ф-ию базового класса
        cout << " " << jump;
    }
};
```

Использовать конструктор базового класса

наследование

```
class Animal
{
protected: // модификатор доступа, который дает доступ к полям age, name в дочернем классе
    int age;
    string name;

public:
    Animal(string name = " ", int age = 0)
    {
        this->name = name;
        this->age = age;
    }

    void show_na()
    {
        cout << name << " " << age;
    }
};

class HomeAn : public Animal
{
    double jump;

public:
    HomeAn(string name = " ", int age = 0, double jump = 0) : Animal(name, age), jump(jump) {}

    void show_home()
    {
        show_na(); //используем ф-ию базового класса
        cout << " " << jump << endl;
    }
};

class WildAn : public Animal
{
    bool roar;

public:
    WildAn(string name = " ", int age = 0, bool roar = 0) : Animal(name, age), roar(roar) {}

    void show_wild()
    {
        show_na();
        cout << " " << roar << endl;
    }
};
```

Класс Animal и его дочерние

наследование

```
int main()
{
    Animal Cat("Barsik", 12);
    HomeAn Puff("Leopold", 10, 5.5);
    WildAn Wolf("Alex", 15, 1);
    Cat.show_na();
    cout <<endl;
    Puff.show_home();
    Wolf.show_wild();
}
```

Main()

наследование

```
int main()
{
    Animal Cat("Barsik", 12);
    HomeAn Puff("Leopold", 10, 5.5);
    WildAn Wolf("Alex", 15, 1);
    Cat.show_na();
    cout <<endl;
    Puff.show_home();
    Wolf.show_wild();
}
```

```
Barsik 12
Leopold 10 5.5
Alex 15 1
```

Main()

Задача

Есть класс 2D точки (координата X и Y). Также есть метод вывода значений координат.

Задача:. С помощью механизма наследования реализуйте наследуемый класс 3D точки (координата X, Y и Z) + метод вывода значений этих координат.

Структуры объектов

Задача

Создайте класс по диаграмме выше.

Задача: в main создайте одного персонажа, введите данные и выведите их. (**На стеке**)

Задача

Создайте класс по диаграмме выше.

Задача: в main создайте одного персонажа, введите данные и выведите их. (**Динамическая память**)

Задача

Создайте класс по диаграмме выше.

Задача: в main создайте **массив** персонажей, сколько потребует пользователь (**на стеке**).

Задача

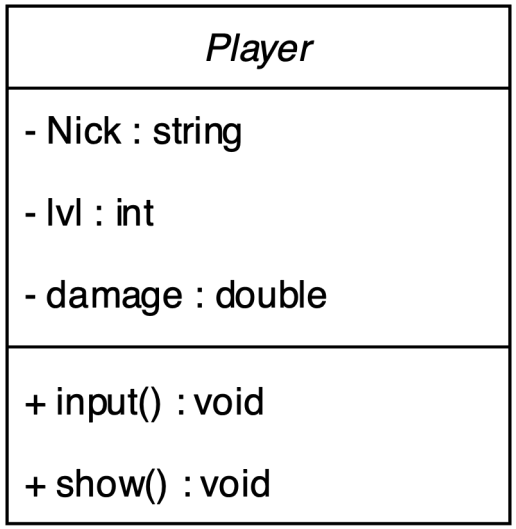


Диаграмма класса

```
int main()
{
    Player arr[100]; // выделяем память под 100 объекта на стеке
    int col;         // кол-во игроков
    cin >> col;

    for (int i = 0; i < col; i++)
    {
        arr[i].input();
    }

    for (int i = 0; i < col; i++)
    {
        arr[i].show();
    }
}
```

Main()

Задача

Создайте класс по диаграмме выше.

Задача: в main создайте **массив** персонажей, сколько потребует пользователь (**динамическая память**).

Абстракция

Что такое Абсстракция?

Где они применяются?

Абстракция используется только на C++?

Абстракция

Абстракция — процесс отвлечения (разделения) от тех или иных характеристик объекта для их избирательного анализа;

Абстракция

Абстракция – это метод программирования (проектирования), основанный на **разделении** интерфейса и реализации.

Абстракция

Плюсы абстракции:

Дочерние классы защищены от непреднамеренных ошибок на уровне пользователя, которые могут повредить состояние объекта.

Реализация дочерних может расширяться или дорабатываться.

Возможность через абстрактный класс создавать объекты и использовать те методы, которые определены типом дочернего потомка.

Задача

Задача: Напишите реализацию класса Rectangle (прямоугольник) с приватными полями длина и ширина. Также реализуйте методы подсчета периметра (per) и площади (sq).

Задача

Задача: Напишите реализацию класса Rectangle (прямоугольник) с приватными полями длина и ширина. Также реализуйте методы подсчета периметра (per) и площади (sq).

```
int main()
{
    Rect kv(5, 5);
    cout << kv.per() << " " << kv.pl() << endl;
}
```

Абстракция

```
class Rect
{
private:
    double weight, height;

public:
    Rect(double weight = 0, double height = 0) : weight(weight), height(height) {}
    void read()
    {
        cin >> weight >> height;
    }
    void show()
    {
        cout << weight << " " << height;
    }

    double per()
    {
        return 2 * (weight + height);
    }

    double pl()
    {
        return (weight * height);
    }
};
```

Класс Rect

Задача

Задача: Напишите реализацию класса Circle (круг) с приватными полями радиус. Также реализуйте методы подсчета длины окружности (per) и площади (sq).

Задача

Задача: Напишите реализацию класса Circle (круг) с приватными полями радиус. Также реализуйте методы подсчета длины окружности (per) и площади (sq).

```
int main()
{
    Circle kr(4.12);
    cout << kr.per() << " " << kr.pl() << endl;
}
```

Абстракция

```
class Circle
{
private:
    double rad;

public:
    Circle(double rad = 0) : rad(rad) {}
    void read()
    {
        cin >> rad;
    }
    void show()
    {
        cout << rad;
    }

    double per()
    {
        return 2 * (PI * rad);
    }

    double pl()
    {
        return (PI * rad * rad);
    }
};
```

Класс Circle

Что общего между классами?

В чем различия?

Можно ли обобщить?

Можем ли мы реализовать механизм наследования?

От Квадрата круг?
От Круга квадрат?

А что если создать общий класс Фигура?

Абстракция

Мы не можем наследовать круг от прямоугольника тк прямоугольник **не частный случай** круга и наоборот.

В данной задаче мы пойдем по пути **абстрагирования** (создание общего класса Figure) от того, кто будет наследником: круг, квадрат, шестиугольник, n – угольник.

В абстрактном классе Figure будут 3 метода: подсчет периметра, площади, деструктор.

ВАЖНО: объект абстрактного класса **не может быть создан**, тк он не является завершенным.

А какая реализация будет у этих методов?

Абстракция

При реализации абстрактного класса часть его методов не будут иметь реализацию

Абстракция

При реализации абстрактного класса часть его методов не будут иметь реализацию

```
class Figure
{
public:
    virtual double pl() = 0;
    virtual double per() = 0;
};
```

Класс Figure с чисто виртуальными функциями

Абстракция

Чисто виртуальная функция (pure virtual function) является функцией, которая объявляется в базовом классе, но не имеет в нем определения.

Поскольку она не имеет определения, то есть тела в этом базовом классе, то всякий производный класс обязан переопределять данную ф-ию.

Для объявления чисто виртуальной функции используется следующая общая форма:

```
virtual тип имя_функции(список параметров) = 0;
```

Абстракция

```
const double PI = 3.14;

class Figure
{
public:
    virtual double pl() = 0;
    virtual double per() = 0;
};

//Чист вирт ф-ии - ф-ии, кот не имеют тела.

class Rect : public Figure
{
private:
    double weight, height;

public:
    Rect(double weight = 0, double height = 0) : weight(weight), height(height) {}

    double per()
    {
        return 2 * (weight + height);
    }

    double pl()
    {
        return (weight * height);
    }
};

class Circle : public Figure
{
private:
    double rad;

public:
    Circle(double rad = 0) : rad(rad) {}

    double per()
    {
        return 2 * (PI * rad);
    }

    double pl()
    {
        return (PI * rad * rad);
    }
};
```

Реализуем механизм наследования

Задача

Задача: Допишите ф-ии ввода и вывода полей.

Абстракция

Для того, чтобы увидеть прелесть абстракции, посмотрим на main().

Через указатель базового класса на объект дочернего класса, будем вызывать тот метод, который нужен объекту.

```
int main()
{
    string what;
    Figure *ptr;
    cin >> what;
    if (what == "Rect")
    {
        Rect re_ptr;
        ptr = &re_ptr;
    }
    if (what == "Cirk")
    {
        Circle ce_ptr;
        ptr = &ce_ptr;
    }

    ptr->read();
    ptr->show();
    cout << "Периметр: " << ptr->per() << " Площадь:" << ptr->pl() << endl;
}
```

Реализация main()

Абстракция

В конечном итоге мы работаем с **абстрактным объектом**, который **сам** понимает на что он ссылается.

```
Rect
2 5
Квадрат: 2 5. Периметр: 14 Площадь:10
```

```
Cirk
4
Круг: 4. Периметр: 25.12 Площадь:50.24
```

Абстракция

```
int main()
{
    string what;
    Figure *ptr;
    cin >> what;
    if (what == "Rect")
    {
        ptr = new Rect;
    }
    if (what == "Cirk")
    {
        ptr = new Circle;
    }

    ptr->read();
    ptr->show();
    cout << "Периметр: " << ptr->per() << " Площадь:" << ptr->pl() << endl;
}
```

Альтернативная реализация main()

Задача

Задача: Добавьте метод compare, который ищет отношение периметра к площади.