Наследование классов в C++

Вопросы

1. Какое ключевое слово используется при создании класса?
2. В чем отличие метода от функции?
3. Что такое конструктор

Наследование

Наследование представляет один из ключевых аспектов объектно-ориентированного программирования, который позволяет наследовать функциональность одного класса (родительского класса) в другом - дочернем классе.

Зачем нужно наследование?

Предположим, у нас есть класс Person с полями name и age и методом Print для вывода информации о человеке:

Зачем нужно наследование?

```
class Person{
   public: void print()  {
      cout << "Name: " << name << "\tAge: " << age << endl;
   }
   string name;      //  имя
   unsigned age;          // возраст
};
```

Зачем нужно наследование?

И класс Employee с полями name, age, company и методом Print для вывода информации о сотруднике:

Зачем нужно наследование?

```
class Employee{
public: void print()    {
      cout << "Name: " << name << "\tAge: " << age << endl;
   }
   string name;      // имя
   unsigned age;         // возраст
   string company;    // компания
};
```

Зачем нужно наследование?

В данном случае класс Employee фактически содержит функционал класса Person: свойства name и age и функцию print.

Зачем нужно наследование?

Мы сталкиваемся с повторением функционала в двух классах. Также мы сталкиваемся с отношением is ("является"). То есть мы можем сказать, что сотрудник компании ЯВЛЯЕТСЯ человеком.

Зачем нужно наследование?

Поэтому в этом случае лучше использовать механизм наследования. Унаследуем класс Employee от класса Person:

```
class Person{
public:
    void print(){
        cout << "Name: " << name << "\tAge: " << age << endl;
    }
    string name;          //  имя
    unsigned age;                 // возраст
};
class Employee : public Person{
public:
    string company;     // компания
};
```

Зачем нужно наследование?

Для установки отношения наследования после названия класса ставится двоеточие, затем идет спецификатор доступа и название класса, от которого мы хотим унаследовать функциональность.

Зачем нужно наследование?

В этом отношении класс Person будет называться базовым классом (также называют суперклассом, родительским классом), а Employee - производным классом (также называют подклассом, классом-наследником).

Зачем нужно наследование?

Воспользуемся этими классами:

```
int main()
{
    Person tom;
    tom.name = "Tom";
    tom.age = 23;
    tom.print();    // Name: Tom        Age: 23

    Employee bob;
    bob.name = "Bob";
    bob.age = 31;
    bob.company = "Microsoft";
    bob.print();    // Name: Bob        Age: 31
}
```

Конструкторы

Конструкторы при наследовании не наследуются.

Если родительский класс содержит только конструкторы с параметрами, то дочерний класс должен вызывать в своем конструкторе один из конструкторов базового класса:

Конструкторы

Добавим конструкторы в класс Person:

```
class Person
{
public:
    string name;          //  имя
    int age;              // возраст
    Person(string n, int a) //конструктор
    {
        name = n;
        age = a;
    }
    void print() const
    {
        cout << "Name: " << name << "\tAge: " << age << endl;
    }

};
```

Конструкторы

Добавим конструкторы в класс Employee:

```
class Employee: public Person
{
public:
    string company;     // компания
    //конструктор вызывает конструктор из родительского класса
    Employee(string name, int age, string c): Person(name, age)
    {
        company = c;
    }

};
```

Конструкторы

Создадим человека и сотрудника:

```
int main() {

    Person person {"Tom", 38};
    person.print();       // Name: Tom          Age: 38

    Employee employee {"Bob", 42, "Microsoft"};
    employee.print();     // Name: Bob          Age: 42

}
```

Конструкторы

После списка параметров конструктора производного класса через двоеточие идет вызов конструктора базового класса, в который передаются значения параметров n и a.

Конструкторы

```
Employee(string name, unsigned age,
string c): Person(name, age)
{
    company = c;
}
```

Запрет наследования

Иногда наследование от класса может быть нежелательно. И с помощью спецификатора final мы можем запретить наследование:

Запрет наследования

```
class Person final
{
};
```

Задание

1. Переписать код со скришотов. (Класс Person и Employee)
2. Добавить класс Student, наследник класса Person.
3. Дополнить класс Student свойствами «специальность» и «курс». Добавить конструктор.
4. Протестировать класс Student.

Задание

1. Переписать код со скришотов. (Класс Person и Employee)
2. Добавить класс Student, наследник класса Person.
3. Дополнить класс Student свойствами «специальность» и «курс». Добавить конструктор.
4. Протестировать класс Student.

Управление доступом

Инкапсуляция

Класс может определять различное состояние, различные функции. Однако не всегда желательно, чтобы к некоторым компонента класса был прямой доступ извне. Для разграничения доступа к различным компонентам класса применяются спецификаторы доступа.

Инкапсуляция

Спецификатор public делает члены класса - поля и методы открытыми, доступными из любой части программы.

```
class Person
{
public:
    string name;          //  имя
    int age;              // возраст
    Person(string n, int a) //конструктор
    {
        name = n;
        age = a;
    }
    void print() const
    {
        cout << "Name: " << name << "\tAge: " << age << endl;
    }

};
```

Инкапсуляция

То есть в данном случае поля name и age и функция print являются открытыми, общедоступными, и мы можем обращаться к ним во внешнем коде. Например, мы можем обратиться к полям класса и присвоить им любые значения, даже если они будут не совсем корректными, исходя из логики программы

Инкапсуляция

```
Person tom("Tom", 22);
tom.name = "";
tom.age = 1001;
```

Инкапсуляция

С помощью спецификатора private мы можем скрыть реализацию членов класса, то есть сделать их закрытыми, инкапсулировать внутри класса. Перепишем класс Person с применением спецификатора private:

```
class Person
{
private:
    string name;          //  имя
    int age;              // возраст
public:
    Person(string n, int a) //конструктор
    {
        name = n;
        age = a;
    }
    void print() const
    {
        cout << "Name: " << name << "\tAge: " << age << endl;
    }

};
```

Инкапсуляция

Теперь мы не можем обратиться к свойствам name и age вне класса Person. Мы можем к ним обращаться только внутри класса Person. А функция print и конструктор по прежнему общедоступные, поэтому мы можем обращаться к ним в любом месте программы.

Задания

1. Сделать поля всех классов закрытыми