

Технология ООП

Санкт-Петербургский государственный политехнический университет

20 сентября 2011

Спецификаторы для данных и методов

```
struct Man
{
char* Name;
int Age;
int age() { return Age; }
};
```

```
Man man;
man.age();
man.Age;
```

Спецификаторы для данных и методов

- private
- public
- protected

```
class Man
{
char* Name;
int Age;

public:
int age() { return Age; }
};

Man man;
man.age();
man.Age; //не будет работать
```

Конструкторы и деструкторы

Конструктор — служит для начальной инициализации объекта при его создании.

Деструктор — вызывается при прекращении жизни объекта, используется для освобождения динамически выделенной памяти.

Носит такое же имя как и класс, не имеет типа возвращаемого значения.

```
class Man
{
char Name[100];
int Age;

public :
Man(char *name) { strncpy(Name, name, 100); }
Man() { strncpy(Name, "test", 100); } //конструктор
    по умолчанию
Man(int age = 100) { Age = age; }
};
```

```
Man::Man(int value1 , int value2)
{
    var1 = value1;
    var2 = 5;
}
```

```
Man::Man(int var1 , int var2) : var1(value1), var2
    (5)
{
}
```

Срабатывает при инициализации объекта класса другим объектом.

```
class Man
{
char *Name;

public :
    Man(const &name) { strncpy(Name, name.Name, 100) }
};
```

Срабатывает при окончании времени жизни объекта. Не имеет параметров.

```
class Man
{
char *Name;

public :
~Man() { delete Name; }
};
```

Можно вызвать деструктор явно `obj-> Man();`

Указатель this

Указатель на текущий объект, используется неявно для обращения к элементам объекта.

```
class Man
{
int Age;

public:
int age() { return Age; }
Man *obj() { return this; }
// int age() { return this->Age; }
};
```

Дружественные функции

Дружественные функции позволяют получить доступ к private членам класса.

```
class Man
{
int Age;
friend void setAge(Man *, int);

public:
void setAge(int age) { Age = age; }
};

void setAge(Man *man, int age)
{
    man->Age = age;
}
```

Дружественные функции

```
class Man;
class User
{
public:
void age(Man &, int);
};
class Man
{
int Age;
friend void User::age(Man &, int);
};
void User::age(Man &man, int age)
{
man.Age = age;
}
```

Дружественный класс

Дружественный класс позволяет сделать все функции класса дружественными.

```
class Man;  
class User  
{  
  public :  
  void age(Man &, int );  
};  
class Man  
{  
  int Age;  
  friend class User;  
};  
void User::age(Man &man, int age)  
{  
  man.Age = age;  
}
```

Статические члены класса

Члены класса для которых память выделяется один раз при создании класса.

```
class Man
{
private:
static int count;
public:
static int getCount() { return count; }
};

int x = Man::getCount();
```

Спецификатор const

Используется для предотвращения изменения данных.

```
class Man
{
private:
int Age;
public:
void age(int _age) const //нельзя изменять члены
    класса
{
Age = _age; //ошибка
}

void change(const Man& man)
{
man.Age = 5; //ошибка
}
};
```

В C++ можно перегрузить любые операции, кроме: '.', '.*', '?:', '::', '#', '##', 'sizeof'.

- Для стандартных типов переопределить операции нельзя.
- Функции-операции не могут иметь аргументов по умолчанию, наследуются за исключением операции '='.
- При перегрузке сохраняется число аргументов, а также приоритет операции.

тип **operator** операция (список параметров) {тело функции}

Перегрузка унарных операций

```
class Man
{
int Age;
Man & operator ++() { ++Age; return *this; }
};

Man man;
++man;
```

```
class Man
{
int Age;
public:
int getAge() { return Age; }
};

bool operator >(const Man &m1, const Man &m2)
{
if(m1.getAge() > m2.getAge())
return true;
return false;
}
```

```
class Man
{
int Age;

Man operator +(const Man &m2)
{
    Man temp;
    temp.Age = Age + m2.Age;
    return temp;
}
};

Man man1, man2;
man1 = man1 + man2;
```

```
class Man
{
int Age;
char *Name;

const Man & operator = (const Man &m1)  { ... }
};
```

Перегрузить для класса массив операцию [].

```
//TODO
```

```
class имя : [private | protected | public]  
    базовый_класс
```

Простое наследование — наследование от одного родителя.

```
class Man
{
char *Name;
int Age;
};

class Stud : public Man
{
int Course;
};

class Prof : public Man
{
char * Dept;
};
```

```
class Man
{
char *Name;
int Age;
};

class Stud : public Man
{
int Course;
};

class Prof : public Man
{
char * Dept;
};
```

Простое наследование

```
class Man
{
    char *Name;
    int Age;
    public:
    int getAge() {return Age;}
};

class Stud : public Man
{
    int Course;
    public:
    int getAge() {return getAge();} //FIXME
    Stud() { printf("%d\n", getAge()); }
};

class Prof : public Man
{
```

Вызов конструкторов и деструкторов при наследовании

Конструкторы вызываются от более старшего класса к младшему, деструкторы наоборот.

```
class A { ... };  
class B : public A { ... };  
class C : public A { ... };  
class D : public A, public B, public C { ... };
```

Объект класса D содержит унаследованные члены класса A в 3х экземплярах.

```
class A { ... };  
class B : virtual public A { ... };  
class C : virtual public A { ... };  
class D : public A, public B, public C { ... };
```

```
Man *man = new Stud;  
man->age();
```

Будет вызван метод класса Man. Механизм раннего связывания.

```
virtual int age() {...}
```

```
Man *man, *man2;  
man = new Stud;  
man2 = new Man;
```

```
man->age();  
man2->age();
```

Механизм позднего связывания.

- Если в базовом классе метод виртуальный, то в производном классе такой же метод становится автоматически виртуальным.
- Виртуальные методы наследуются.
- Нельзя объявить как `static`.
- Можно определить виртуальный метод, как «чисто виртуальный» присвоив ему значение 0.

Деструктор базового класса как правило всегда следует делать виртуальным, чтобы избежать утечек памяти.

Абстрактный класс — класс имеющий «чисто виртуальные» функции, объект такого класса не может быть создан, а только унаследован.