Технологии и методы программирования

Часть 3

Ст. преподаватель
кафедры ПИВТ
Воронцова И.О.

2020 год

Наследование

Создание класса-наследника в C++

```
// базовый класс
class A
{
  ...
} ;
//класс-наследник
class B : public A
{
  ...
} ;
```

Пример базового класса

```
class Person
{
protected:
   char name[30];
public:
   Person ();
   void setName (char* );
   void show ();
} ;
```

Класс-наследник

```
class Student : public Person
{
    char dept[30];
public:
    Student (char*, char* );
    void setDept (char* );
    void show ();
} ;
```

Методы класса Person

```
Person::Person()
{
    strcpy (name, "Noname");
}

void Person::setName (char* n)
{
     strcpy (name, n);
}

void Person::show ()
{
std::cout << "My name is " << name << std::endl;
}
```

Собственные методы класса Student

```
Student::Student (char* n, char* d)
{
    strcpy (name, n);
    strcpy (dept, d);
}
void Student::setDept (char* d)
{
    strcpy (dept, d);
}
void Student::show ()
{
    std::cout << name << “  “ << dept << std::endl;
}
```

Текст программы

```
int main ()
{
    Person a;
    a.setName ("Tom");
    Student b("Ann", "MTS");
    a.show ();
    b.show ();
    Person* pperson = &b;
    Student* pstudent = &b;
    pperson->show ();
    pstudent->show ();
    b.setName ("Kate");
    b.setDept ("GF");
    pstudent->show ();
    return 0;
}
```

На экране:

My name is Tom

Ann MTS

My name is Ann

Ann MTS

Kate GF

Виртуальная функция

```
class Person
{
    char name[30];
public:
    Person (char* );
    void setName (char* );
    virtual void show ();
} ;
```

Текст программы

```
int main ()
{
    Person a;
    a.setName ("Tom");
    Student b("Ann", "MTS");
    a.show ();
    b.show ();
    Person* pperson = &b;
    Student* pstudent = &b;
    pperson->show ();
    pstudent->show ();
    b.setName ("Kate");
    b.setDept ("GF");
    pstudent->show ();
    return 0;
}
```

На экране:

My name is Tom

Ann MTS

Ann MTS

Ann MTS

Kate GF

Полиморфизм

Статический полиморфизм — поддерживается посредством перегрузки функций и операторов во время компиляции.

Динамический полиморфизм — поддерживается посредством виртуальных функций во время выполнения программы.

Конструкторы и наследование

```
class Person
{
    protected:
        char name[30];
    public:
        Person (char*); //конструктор с параметром
        void setName (char* );
        void show ();
} ;
```

Реализация конструкторов

```
Person::Person (char* n)
{
    strcpy (name, n);
}

Student::Student (char* n, char* d) : Person (n)
{
    strcpy (dept, d);
}
```

Деструкторы и наследование

```
class A {
protected:
    int a;
public:
    A()  { a=5; cout<<"Create A"<<endl;}
    virtual ~A()  { cout<<"Delete A"<<endl;}
    void setA(int pa)  { a=pa; }
    int getA()  { return a; }
} ;
```

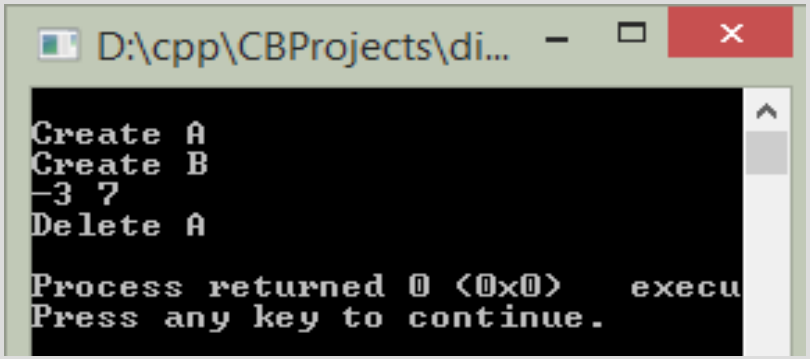
Деструкторы и наследование (2)

```
class B:public A{
protected:
    int b;
public:
    B():b(7)  {  cout<<"Create B"<<endl; }
    virtual ~B()  { cout<<"Delete B"<<endl; }
    void setB(int pb) { b=pb; }
    int getB()  { return b; }
    void print() { cout<<a<<" "<<b<<endl; }
};
```

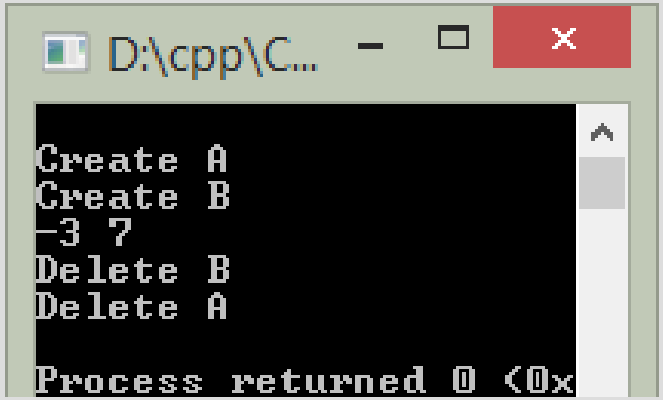
Деструкторы и наследование (3)

```
int main()
{
    A* fobj = new B;
    fobj->setA(-3);
    ((B*) fobj) -> print();
    delete fobj;
    return 0;
}
```

Невиртуальный деструктор



Виртуальный деструктор



Множественное (прямое) наследование

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

class A {
protected:
    int a;
public:
    A() { a=5; }
   virtual ~A() {}
    void setA(int pa){ a=pa;}
    int getA(){ return a;}
} ;
```

```
class B {
protected:
    int b;
public:
    B():b(7){}
    virtual ~B() {}
    void setB(int pb){ b=pb;}
    int getB(){ return b;}
};
```

Множественное наследование (продолжение)

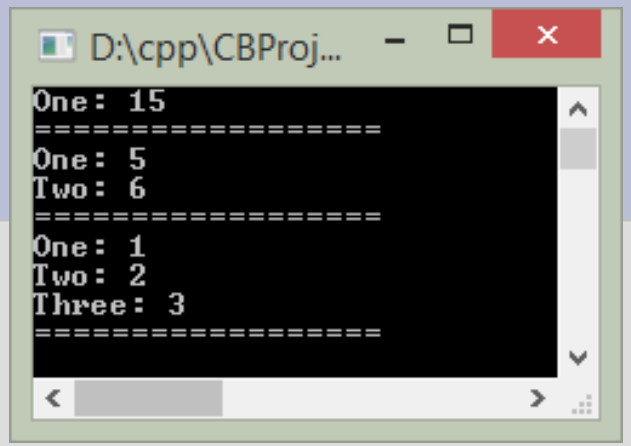
```
class C:public A, public B {
public:
    virtual ~C() {}
    void printAll()  {  cout<<a<<"  "<<b<<endl; }
};
int main()
{
    C object;
    object.printAll();
    object.setA(-3);
    object.setB(-18);
    object.printAll();
    return 0;
}
```

На экране:
5 7
-3 -18

Иерархия классов

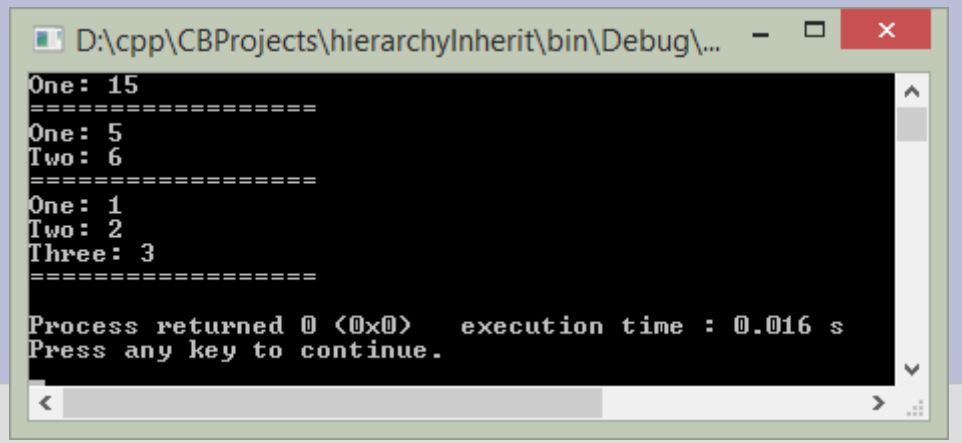
```
class One {
protected:
    int a;
public:
    One(int pa) { a=pa; }
    virtual ~One() {}
    virtual void print() { cout <<"One: "<<a<<endl; }
} ;
class Two: public One {
protected:
    int b;
public:
    Two(int pa,int pb) : One(pa) { b=pb; }
    virtual ~Two() {}
    virtual void print() { One::print(); cout <<"Two: "<<b<<endl; }
};
```

Иерархия классов
(продолжение)




```
class Three: public Two {
protected:
    int c;
public:
    Three(int pa,int pb,int pc): Two(pa,pb) { c=pc; }
    virtual ~Three() {}
    virtual void print() { Two::print(); cout <<"Three: "<<c<<endl; }
};
int main() {
    One obj1(15), *ptr=&obj1;
    ptr->print(); cout<<"====================\n";
    Two obj2(5,6); ptr=&obj2;
    ptr -> print(); cout<<"====================\n";
    Three obj3(1,2,3); ptr=&obj3;
    ptr->print(); cout<<"====================\n";
    return 0;
}
```

Иерархия классов




```
class One {
protected:
    int a;
public:
    One(int pa){ a=pa;}
    virtual ~One() {}
    virtual void print() {
        cout <<"One: "<<a<<endl;
    }
} ;

class Two: public One {
protected:
    int b;
public:
    Two(int pa,int pb) : One(pa) {b=pb;}
    virtual ~Two() {}
    virtual void print() {
        One::print();
        cout <<"Two: "<<b<<endl;
    }
};
```

```
class Three: public Two  {
protected:
    int c;
public:
    Three(int pa,int pb,int pc):Two(pa,pb) {c=pc;}
    virtual ~Three() {}
    virtual void print() { Two::print();
    cout <<"Three: "<<c<<endl;
    }
};
int main()  {
    One obj1(15), *ptr=&obj1;
    ptr->print(); cout<<"==========\n";
    Two obj2(5,6); ptr=&obj2;
    ptr->print(); cout<<"==========\n";
    Three obj3(1,2,3); ptr=&obj3;
    ptr->print(); cout<<"==========\n";
    return 0;
}
```

Абстрактный класс

```
#ifndef SHAPE_H
#define SHAPE_H

class Shape // объявление абстрактного класса
{
public:
      //  Shape();
        virtual ~Shape() {}
        virtual float getSquare()=0;    //функция
        virtual void getInfo()=0;         // не реализована
protected:
private:
};

#endif // SHAPE_H
```

Абстрактный класс (2)

```
#ifndef CIRCLE_H
#define CIRCLE_H
#include "Shape.h"

class Circle:public Shape //создание класса-наследника
{ // базовый класс - абстрактный
public:
        Circle();
        virtual ~Circle() { };
        float getSquare();
        void getInfo();
private:
        int r;
        int xc;
        int yc;
};
#endif // CIRCLE_H
```

Абстрактный класс (3)

```
#include "..\include\Circle.h"
#include <cmath>
#include <iostream>
using namespace std;

Circle::Circle():r(10),xc(15),yc(15)   { }

float Circle::getSquare()
{
    return M_PI*r*r;
}
void Circle::getInfo()
{
    cout<<"Circle:" << r << " " << xc << " " << yc <<endl;
}
```

Абстрактный класс (4)

```
#ifndef RECT_H
#define RECT_H
#include "Shape.h"

class Rect:public Shape //создание класса-наследника
{ // базовый класс - абстрактный
public:
	Rect();
	virtual ~Rect()  { };
	float getSquare();
	void getInfo();
private:
	int x1;
	int y1;
	int x2;
	int y2;
};
#endif // RECT_H
```

Абстрактный класс (5)

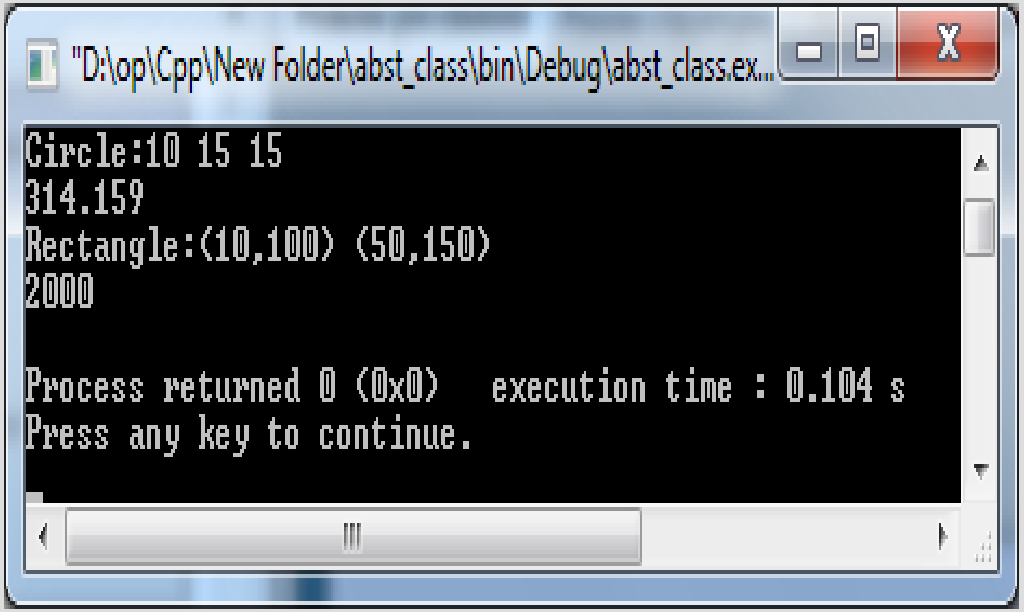
```
#include "..\include\Rect.h"
#include <iostream>
using namespace std;

Rect::Rect()  { x1=10; y1=100; x2=50; y2=150; }

float Rect::getSquare()
{
    return (x2-x1)*(y2-y1);
}
void Rect::getInfo()
{
    cout<<"Rectangle:(" << x1<< ","<< y1<<") (" ;
    cout << x2 << "," << y2 << ")"<<endl;
}
```

Абстрактный класс (6)

```
#include <iostream>
#include "include/Shape.h"
#include "include/Rect.h"
#include "include/Circle.h"
using namespace std;
int main()
{
    Circle s1;
    Rect s2;
    Shape* mas[2];
    mas[0]=&s1;
    mas[1]=&s2;
    for(int i=0;i<2;i++)
    {
        mas[i]->getInfo();
        cout<< mas[i]->getSquare()<<endl;
    }
    return 0;
}
```



Язык UML

UML: Unified Modeling Language.
Стандарты: 1.0, 2.0
Назначение: графический язык для визуализации, конструирования и документирования систем, в т.ч. программного обеспечения.
Разработчики: Грейди Буч, Джеймс Рамбо, Айвар Джекобсон.
Элементы языка: диаграммы и их компоненты (предметы, отношения).

Диаграммы UML

Структурные диаграммы:

- диаграмма компонентов,
- диаграмма классов,
- диаграмма объектов,
- диаграмма развертывания.

Диаграммы поведения:

- диаграмма прецедентов,
- диаграмма последовательности,
- диаграмма состояний.

Диаграмма классов

Обозначение класса на UML-диаграммах:

Полное обозначение

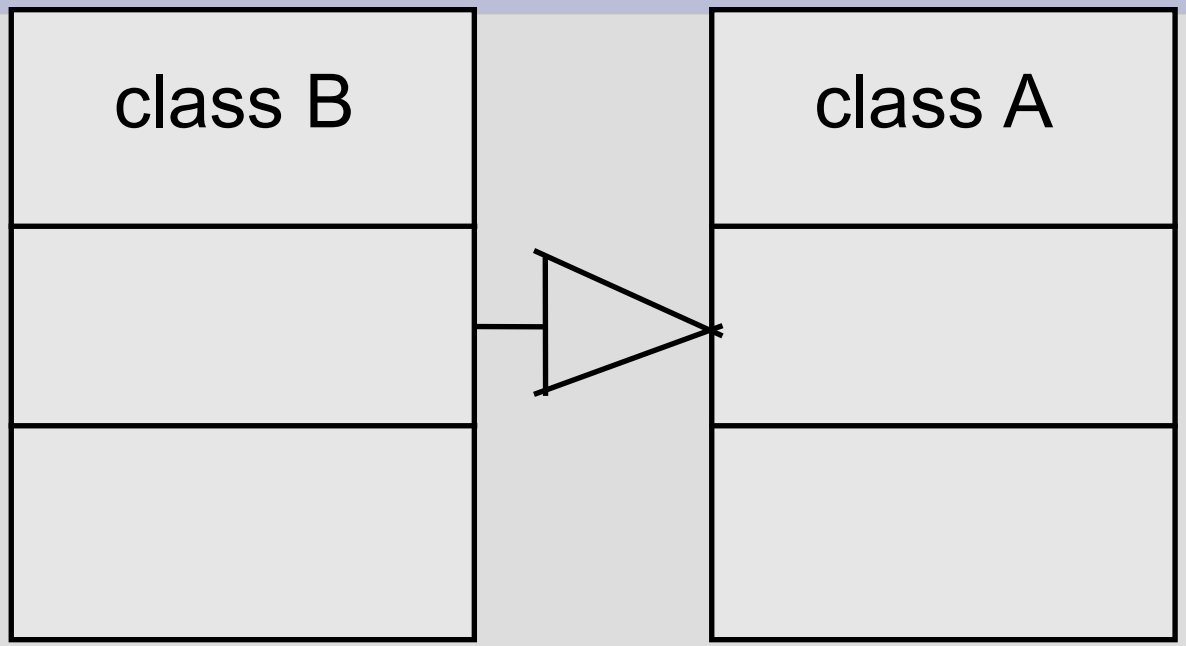
Упрощенное обозначение

Обозначение доступности члена класса:
- private
+ public
# protected

Взаимодействие классов

- Виды взаимодействия:
  - Наследование
  - Композиция
  - Агрегация
  - Зависимость (клиент-серверное взаимодействие)
  - Ассоциация

Наследование



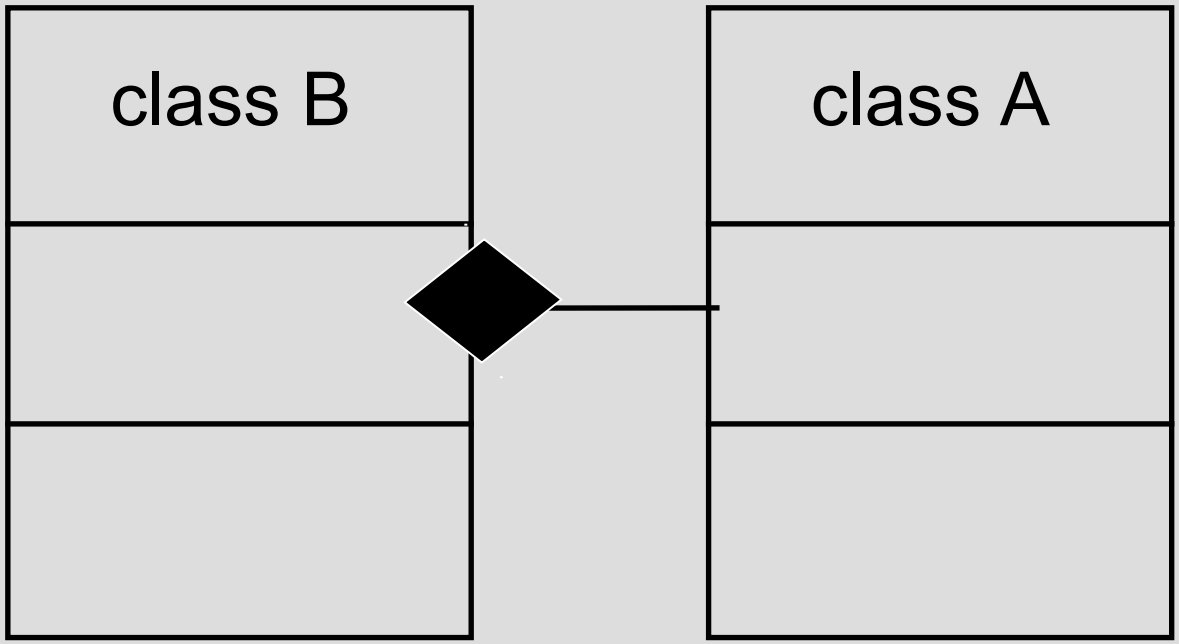

```
class A
{
// . . .
} ;

class B : public A
{
// . . .
} ;
```

Класс-наследник

Базовый класс

Композиция



Внешний класс

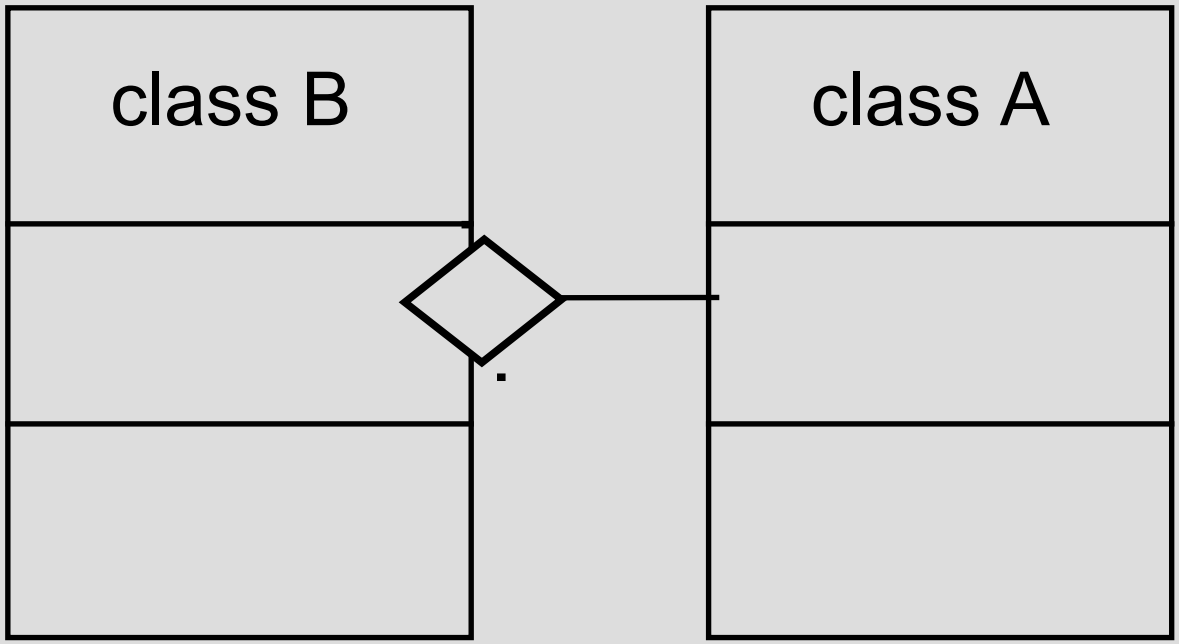
Внутренний класс

```
class A
{
// . . .
public:
      void Message ( );
} ;

class B
{
// . . .
      A a;
} ;

int main ( )
{
      B b ;
// . . .
      b . a . Message (  );
// . . .
}
```

Агрегация



Внешний класс

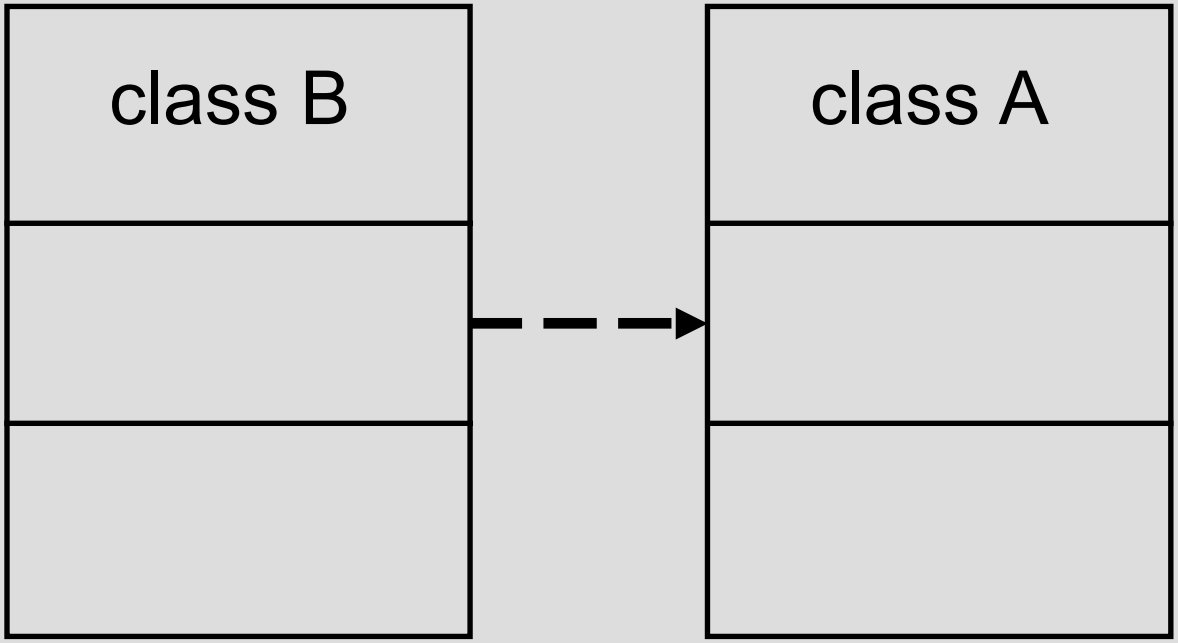
Внутренний класс

```
class A
{
// . . .
public:
      void Message ( );
} ;

class B
{
// . . .
      A* pa;
} ;

int main ( )
{
      A a;
      B b ;
      b.pa =  &a;
// . . .
      b . pa-> Message (  );
// . . .
}
```

Зависимость



Клиент

Сервер

```
class A
{
// . . .
} ;
```

```
class B
{
// . . .
public:
      void Fn ( A*) ;
} ;

int main ( )
{
      A a;
      B b ;
// . . .
      b . Fn ( &a );
// . . .
}
```

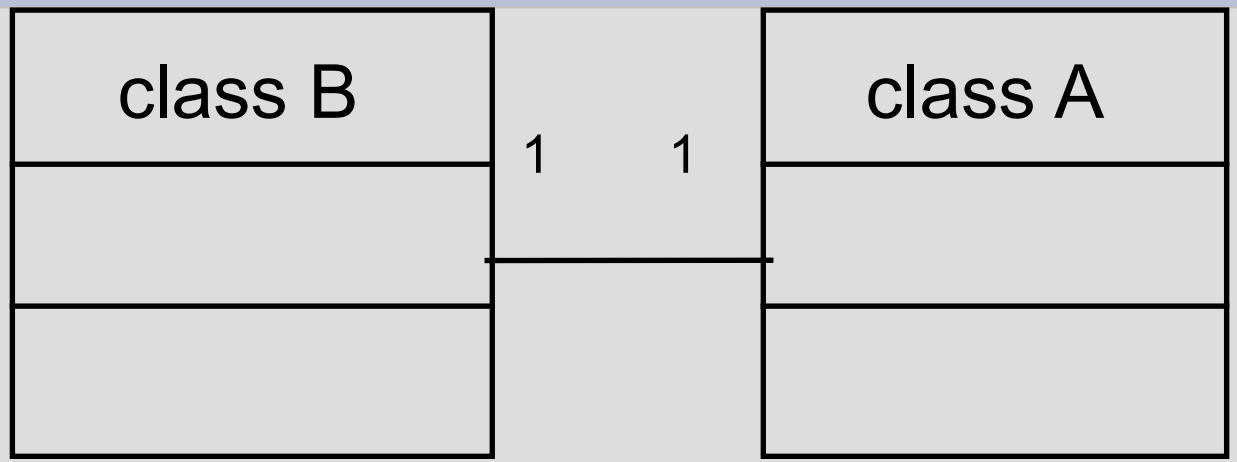
Ассоциация

Мощность связи:

Один-к-одному

Один-ко-много

Много-ко-много



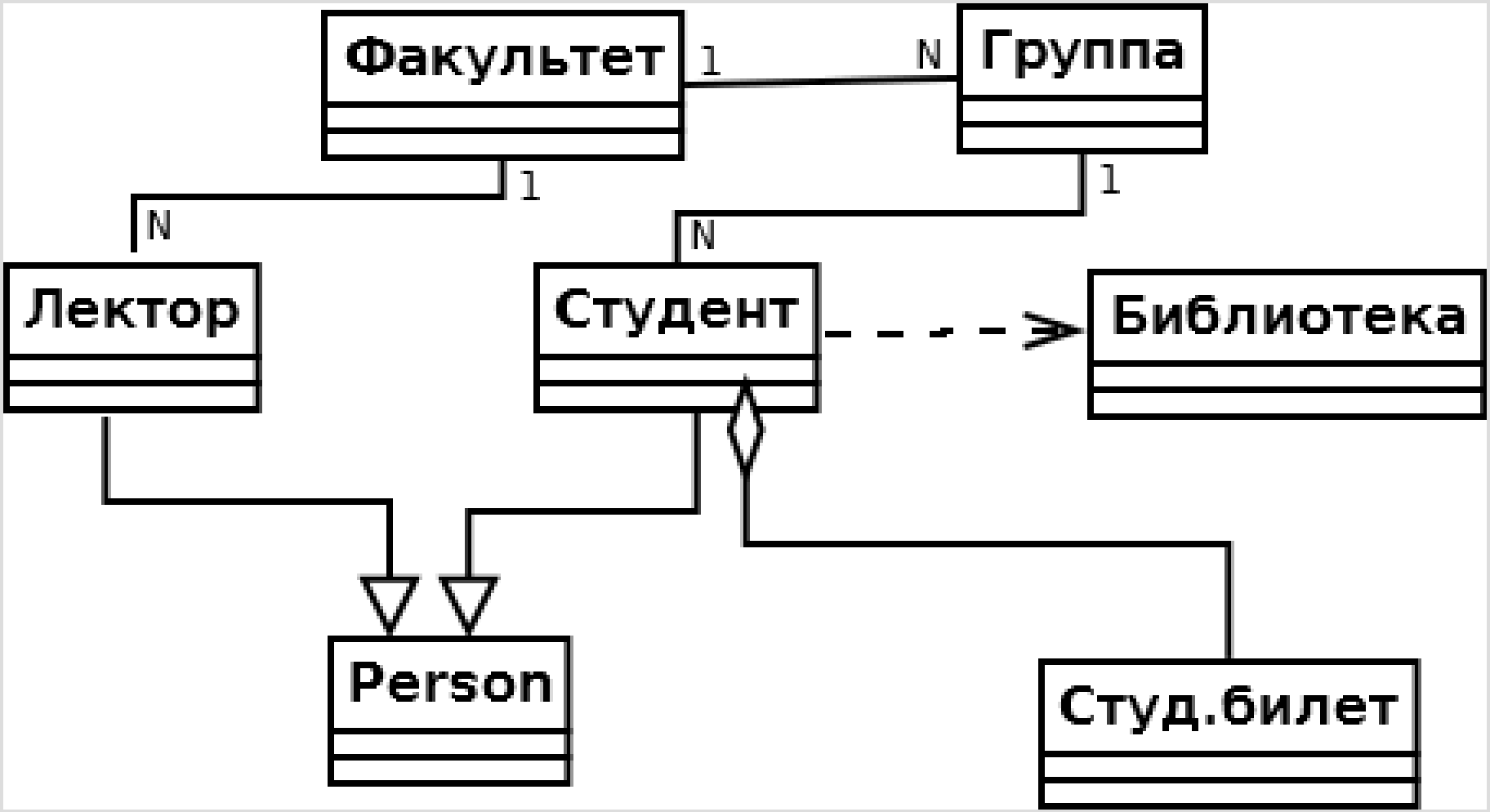
Пример связи один-к-одному

```
class A;
class B
{
// . . .
public:
A* pa;
} ;
```

```
class A
{
// . . .
public:
B* pb;
} ;
```

```
int main ( )
{
      A a;
      B b ;
// . . .
      b.pa = &a;
      a.pb = &b;
// . . .
}
```

Пример диаграммы классов



Друзья класса

```
class First
{
        int x;
    public:
        int getX()  {  return x;  }
        void setX (int);
        friend class Second;
        friend void set_X (First&, int);
} ;
void First::setX (int xx)
{
        x = xx;
}
```

Друзья : глобальная функция или класс

```
class Second
{
// . . .
public:
     void setX (First&, int);
} ;
void Second::setX (First& f,
    int xx)
{
     f.x = xx;
}
void set_X (First& f, int xx)
{
     f.x = xx;
}
```

```
int main ()
{
     First first;
     first.setX (10);
     cout << first.getX () << endl;
     Second second;
     second.setX (first, 20);
     cout << first.getX () << endl;
     set_X (first, 30);
     cout << first.getX () << endl;
}
```

На экране:

- 10
- 20
- 30

Перегрузка операторов

Встроенные перегружаемые операторы:

= == != < > <= >= + - * / ++(инкремент)

- -(декремент) <<(вывод) >>(ввод) += -=

Операторные функции:

<возвр_тип> operatorX(<тип_парам>);

Глобальная функция:

MyClass operator+(MyClass&,MyClass&);

Функция-член класса:

MyClass MyClass::operator+(MyClass&);

Встроенные неперегружаемые операторы:

. :: ?: (оператор условия)

Глобальные операторные функции

```
bool operator==(MyClass, MyClass); // != < >

MyClass operator+(MyClass&, AnyType); // - * /

istream& operator>> (istream&, MyClass&);
ostream& operator<< (ostream&,const MyClass&);
```

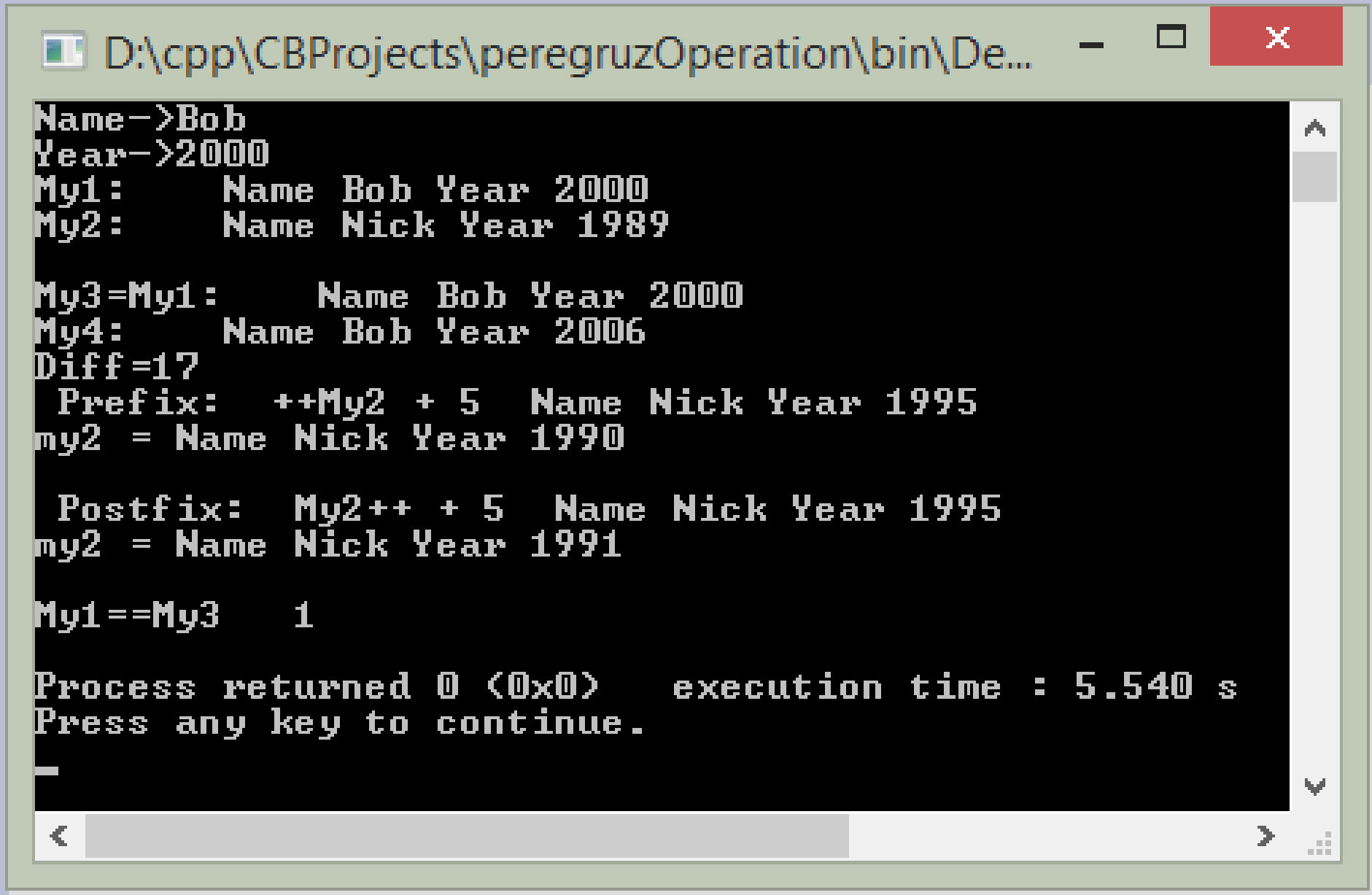
Функции-члены класса

```
bool operator==(MyClass); // != < >

MyClass operator+(AnyType); // - * /

MyClass& operator++(); // - -  (++a)
MyClass operator++(int); // - -  (a++)
MyClass& operator=(MyClass&);
MyClass& operator+=(AnyType); // -=
```

Результат работы программы



person.h

```
#pragma once
#include <iostream>

class Person {
    char name[20];
    int year;
public:
    Person(void);
    Person(char* n, int y);
    bool operator==(Person&);         // оператор отношения
    Person operator+(int);            // сложение
    int operator-(Person&);   //вычитание (результат - целое число)
    Person& operator++();             // префиксный инкремент
    Person operator++(int);           // постфиксный инкремент
    Person& operator=(Person&);    //присваивание
    // Перегрузка ввода через объект класса istream (cin)
    friend std::istream& operator>>(std::istream&,Person&);
    // Перегрузка вывода через объект класса ostream (cout)
    friend std::ostream& operator<<(std::ostream&,const Person&);
};
```

main.cpp

```
#include "person.h"

int main(int argc, char* argv[])
{
    Person my1, my2("Nick",1989);
    std::cin >> my1;
    std::cout << "My1:    " << my1;
    std::cout <<  "My2:    " << my2 <<std::endl;
    Person my3;
    my3=my1;
    std::cout <<  "My3=My1:    " << my3;
    Person my4=my3+6;
    std::cout <<  "My4:    " << my4;
    int diff=my4-my2;
    std::cout << "Diff=" << diff << std::endl;
  std::cout <<"Prefix: ++My2 + 5 " << ++my2 + 5<< "my2 ="<<my2<<endl;
std::cout <<"Postfix: My2++ + 5 " << my2++ + 5<< "my2 = "<<my2<<endl;
    std::cout << "My1==My3   " << (my1==my3) <<std::endl;
    return 0;
}
```

person.cpp (1)

```
#include ".\person.h"
#include <string.h>

Person::Person(void)
{
    strcpy(name, "Noname");
    year=0;
}
Person::Person(char* n, int y)
{
    strcpy(name, n);
    year=y;
}
bool Person::operator==(Person& p)
{
     bool cmp =!strcmp(name,p.name)&&year==p.year;
     return cmp;
}
```

person.cpp (2)

```
Person Person::operator+(int y)
{
    Person temp;
    temp.year= year+y;
    strcpy(temp.name,name);
    return temp;
}

int Person::operator-(Person& p)
{
    int temp;
    temp=year-p.year;
    return temp;
}
```

person.cpp (3)

```
Person& Person::operator++()
{
    year++;
    return *this;
}

Person Person::operator++(int)
{
    Person ps(*this);
    ++year;
    return ps;
}
```

person.cpp (4)

```
Person& Person::operator=(Person& p)
{
    if (this==&p)
    return *this;
    strcpy(name, p.name);
    year=p.year;
    return *this;
}
```

person.cpp (4)

```
std::istream& operator>>(std::istream& is,Person& p)
{
    char buf[20]; int y;
    std::cout << "Name->";
    is>>buf;    strcpy(p.name,buf);
    std::cout << "Year->";
    is >> y;    p.year = y;
    return is;
}

std::ostream& operator<<(std::ostream& os,const Person& p)
{
    os << "Name " << p.name << " Year " << p.year << std::endl;
    return os;
}
```