ОБЕКТНО-ОРИЕНТИРАНО ПРОГРАМИРАНЕ

Магдалина Тодорова

спец. Компютърни науки, I курс, I поток

ФМИ, СУ „Св. Климент Охридски“
2017/2018

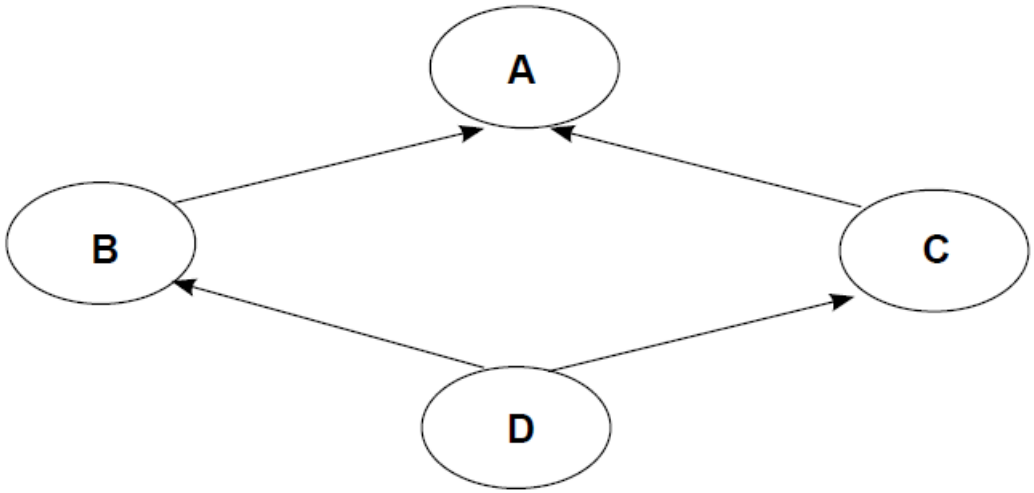
Тема № 14

ВИРТУАЛНИ КЛАСОВЕ

Виртуални класове

При реализиране на йерархии на класове с множествено наследяване е възможно един производен клас да наследи *многократно* даден базов клас.

Пример:



Виртуални класове

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

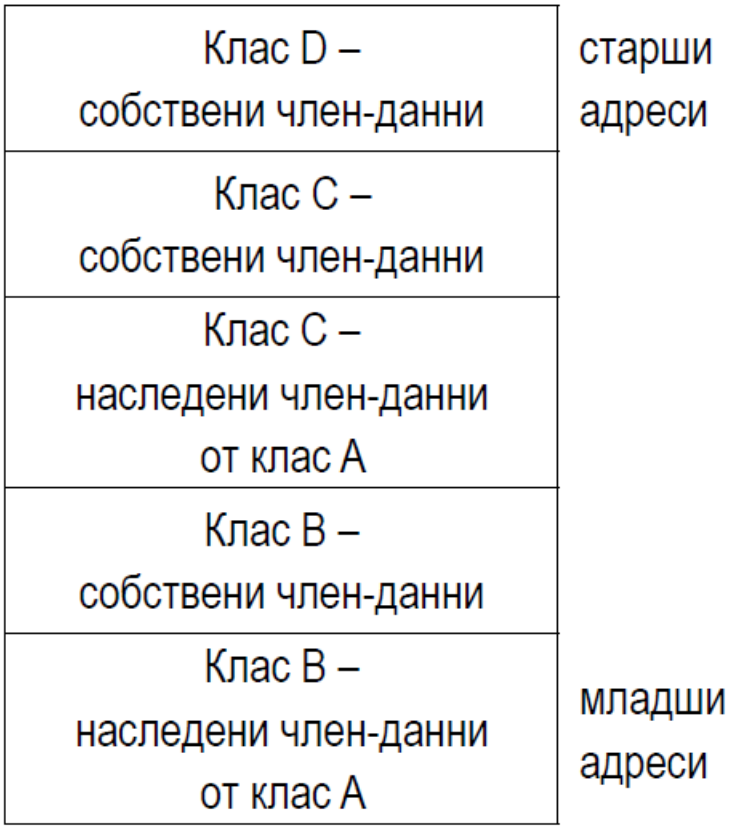
Виртуални класове

Класът *D* индиректно наследява класа *A* *двукратно*.

На пръв поглед за член-функциите двойното наследяване не е от значение, тъй като за всяка член-функция се съхранява само едно копие.

Член-данните обаче се дублират и обект на класа *D* ще наследи двукратно всяка член-данна, декларирана в класа *A*. Обект на класа *D* в паметта ще има вида:

Виртуални класове



Виртуални класове

Този пример илюстрира един от недостатъците на многократното наследяване на клас – *неефективността от поддържането на множество копия на наследени член-данни*.

Ще покажем още недостатъци на многократното наследяване на класове, свързани с член-функциите.

Виртуални класове

```
class A
{ public:
      A(int a)
      { x = a;
      }
      int f() const;
      void print() const;
      int x;
};

int A::f() const
{ return x;
}
void A::print() const
{ cout << "A::x " << x << endl;
}
```

Виртуални класове

```
class B : public A
{ public:
      B(int a, int b) : A(a)
      { x = b;
      }
      int f() const;
      void print() const;
      int x;
};
int B::f() const
{ return x;
}
void B::print() const
{ A::print();
  cout << "B::x " << x << endl;
}
```

Виртуални класове

```
class C : public A
{ public:
      C(int a, int c) : A(a)
      { x = c;
      }
      int f() const;
      void print() const;
      int x;
};
int C::f() const
{ return x;
}
void C::print() const
{ A::print();
  cout << "C::x " << x << endl;
}
```

Виртуални класове

```
class D : public B, public C
{ public:
      D(int a, int b, int c, int d) : B(a, b), C(c, d)
      { }
      void func() const;
      void print() const;
};

void D::print() const
{ B::print();
  C::print();
}
```

Виртуални класове

В резултат на дефиницията

D d(1, 2, 3, 4);

се получава ***нееднозначност***: наследената двукратно член-данна *x* на класа *A* има две различни стойности 1 и 3. Този проблем частично може да се реши като конструкторът на класа *D* се дефинира по следния начин:

```
D(int a, int b, int c) : B(a, b), C(a, c)
{}
```

Виртуални класове

Друг проблем възниква при опит в член-функции на класа *D* да се използват двукратно наследените компоненти *A::x*, *A::f()* или *A::print()*.

Пример. В резултат от компилирането на функцията

```
void D::func() const
{ A::print();
}
```

се получава грешката

... error C2385: 'D::A' is ambiguous

Виртуални класове

Причината е следната:

Аргументът на *func* е *this*, който е от тип *const D**.

Чрез него в тялото на *func* се активира член-функцията *print* на класа *A*. Указателят *this* сочи обект на класа *D*, който съдържа два обекта на базовия клас *A*. Кой от тях да се свърже с извиканата функция *print*?

Виртуални класове

Ако е необходим само **достъп** до „конфликтните“ наследени компоненти на класа *A* (без да се променят стойностите), осъществяването му става чрез последователно прилагане на операцията за явно преобразуване на типове.

При атрибут за област *public*, обект на производен клас може да се преобразува в обект на основен клас с неявни преобразувания.

Виртуални класове

Поради двата клона в йерархията, обект от клас *D* не може да се преобразува директно в обект от клас *A*. За обекта *d*, дефиниран по-долу

D d(1, 2, 3, 4);

са възможни следните последователни преобразувания:

(A)(B) d;

(A)(C) d;

Виртуални класове

Пример.

```
void D::func() const
{ cout << "Derived member x in a part A-B-D "
       << ((A)(B)*this).x << endl
       << "Derived member x in a part A-C-D "
       << ((A)(C)*this).x << endl
       << "Derived member-function f() in a part A-B-D "
       << ((A)(B)*this).f() << endl
       << "Derived member-function f() in a part A-C-D "
       << ((A)(C)*this).f() << endl
       << "Derived member-function f() in part C-D "
       << ((C)*this).f() << endl
       << "Derived member-function f() in part B-D "
       << ((B)*this).f() << endl;
}
```

Виртуални класове

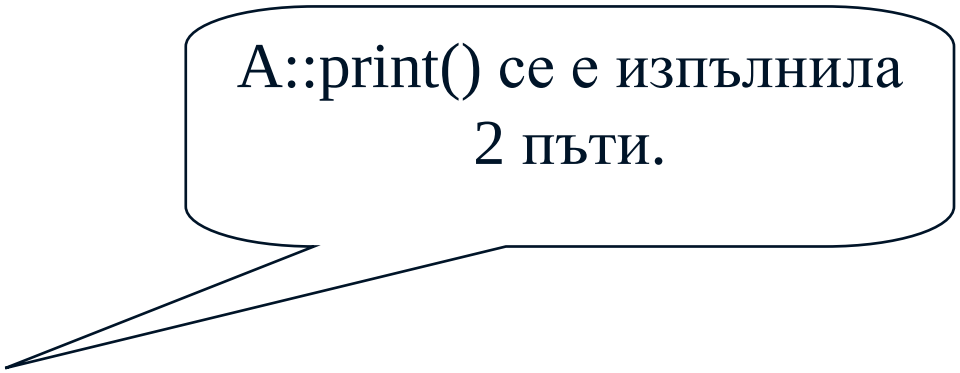
Още един недостатък на многократното наследяване.

Пример. Резултатът от изпълнението на фрагмента

```
D d(1, 2, 3, 4);
d.print();
```

e

A:: x 1
B:: x 2
A:: x 3
C:: x 4



Виртуални класове

Преодоляването на голяма част от недостатъците на многократното наследяване на клас се осъществява чрез използване на т.н.

виртуални основни класове.

Виртуални класове

Чрез виртуалните основни класове се дава възможност да се „поделят” основни класове. Когато един клас е виртуален, се създава само едно негово копие. В нашия случай, ако класът *A* се определи като виртуален за класовете *B* и *C*, класът *D* ще съдържа само един „поделен“ основен клас *A*.

Декларацията на основен клас като виртуален се осъществява като в декларацията на производния клас заедно с името и атрибута за област на основния клас се укаже и ключовата дума *virtual*.

Виртуални класове

Пример.

```
class A
{ public:
      A(int a)
      { x = a;
      }
      int f() const;
      void print() const;
      int x;
};
int A::f() const
{ return x;
}
void A::print() const
{ cout << "A::x " << x << endl;
}
```

Виртуални класове

```
class B : virtual public A
{ public:
      B(int a, int b) : A(a)
      { x = b;
      }
      int f() const;
      void print() const;
      int x;
};
int B::f() const
{ return x;
}
void B::print() const
{ A::print();
  cout << "B::x " << x << endl;
}
```

Виртуални класове

```
class C : virtual public A
{ public:
      C(int a, int c) : A(a)
      { x = c;
      }
      int f() const;
      void print() const;
      int x;
};
int C::f() const
{ return x;
}
void C::print() const
{ A::print();
  cout << "C::x " << x << endl;
}
```

Виртуални класове

```
class D : public B, public C
{ public:
      D(int a, int b, int c, int d) : A(a), B(a, b),
                                      C(c, d)
      {}
      void func() const;
      void print() const;
};

void D::print() const
{ B::print();
  C::print();
}
```

Виртуални класове

Особености на виртуалните класове

1) ***при дефинирането на конструкторите на наследените класове***

Нека *A* е виртуален основен клас за класа *B*, класът *B* е основен за класа *D*, който пък е основен за класа *E*. Ако класът *A* има конструктор с параметри и няма подразбиращ се конструктор, конструкторът с параметри на *A* трябва да бъде извикан не само от конструктора на класа *B*, но и от конструкторите на класовете *D* и *E*.

Виртуални класове

Особености на виртуалните класове

Правило: ***Конструкторите с параметри на виртуални класове трябва да се извикват от конструкторите на всички класове, които са техни наследници, а не само от конструкторите на преките им наследници.***

Виртуални класове

2) ***промяната на реда на инициализиране при изпълнение на конструкторите***

✓Инициализирането на виртуалните основни класове предхожда инициализирането на другите основни класове в декларацията на производния клас. Ако производен клас наследява невиртуален и виртуален клас, конструкторът на виртуалния клас се изпълнява преди конструктора на невиртуалния клас.

✓При наличие на няколко виртуални класа извикването на конструкторите става съгласно реда им в декларацията на производния клас.

Виртуални класове

3) ***Изпълнението на обръщението към конструктора на виртуалния основен клас е еднократно.***

Пример.

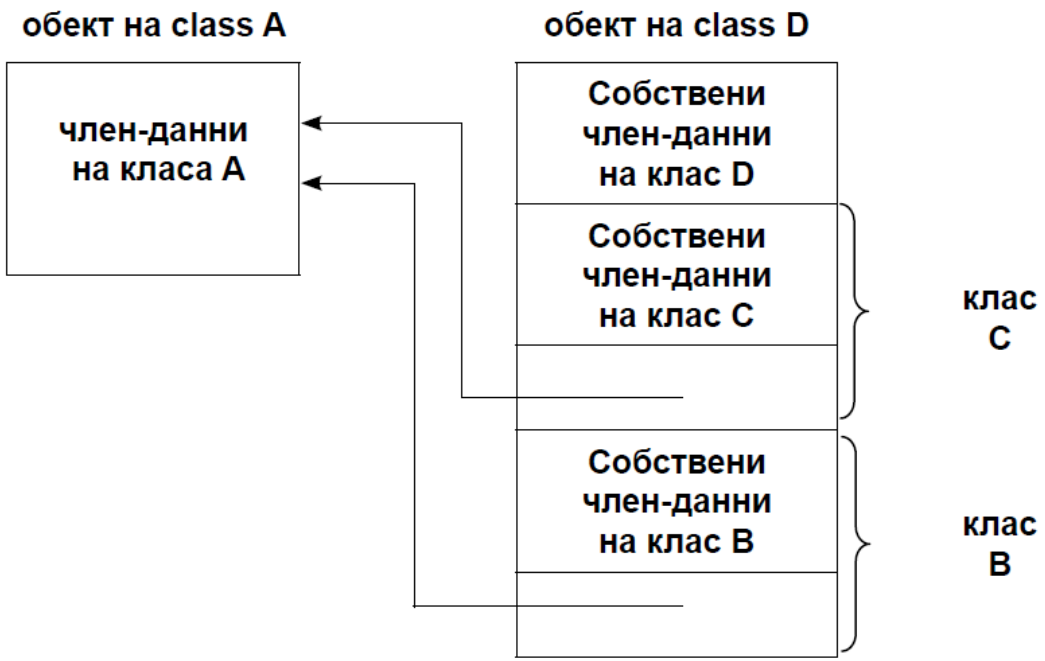
D d(1,2,3,4);

- A(1)
- B(1, 2) A(1) не се изпълнява
- C(3, 4) A(3) не се изпълнява

Виртуални класове

Как използването на виртуални основни класове преодолява недостатъците на многократното наследяване?

Виртуални класове



Виртуални класове

Примери. Допустими са

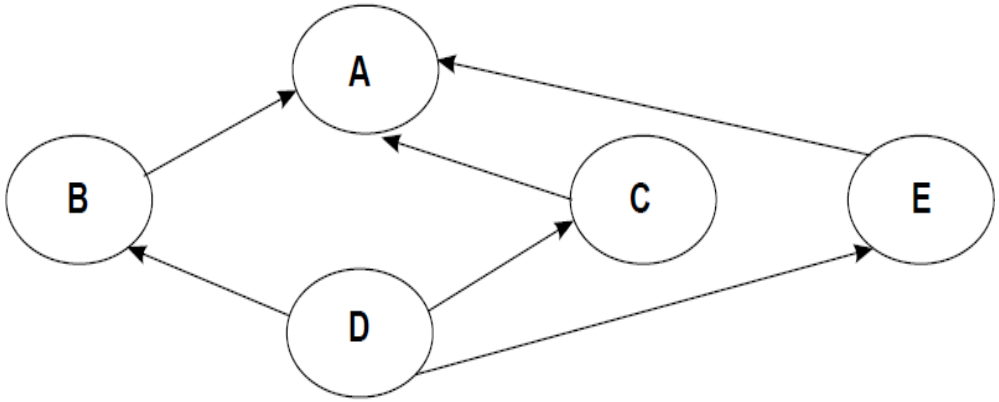
```
void D::func() const
{ cout << A::x << endl;
}

void D::func() const
{ A::print();
}
```

Виртуални класове

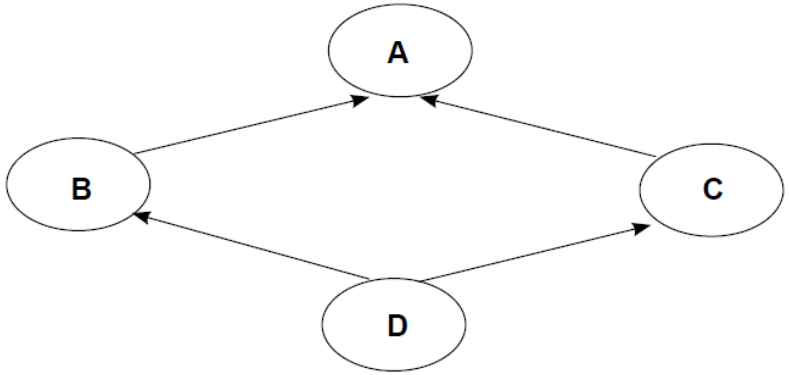
Характеристиките на виртуалните класове могат да се комбинират с тези на невиртуалните.

Например, в йерархията



класът *A* е виртуален за класовете *B* и *C* и не е виртуален за класа *E*. Има НЕЕДНОЗНАЧНОСТ!!!
Преодолява се с преобразувания.

Правило!



класът *A* е виртуален за класовете *B* и *C*, но атрибутът му за област е *public* за класа *B* и *private* за класа *C*.

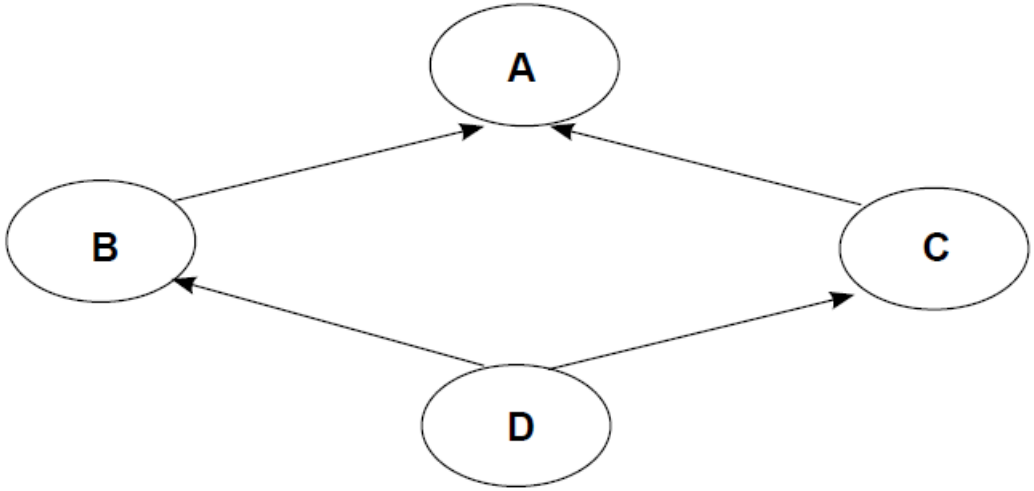
Правило:

ако в някоя декларация виртуалният клас е обявен като public се счита, че той е с атрибут public във всички други негови декларации като виртуален основен клас.

Виртуални класове

Задача.

Да се дефинира йерархията



като класовете *A*, *B*, *C* и *D* имат член-данна *x* от тип символен низ, която е реализирана в динамичната памет. За всеки от класовете да се дефинира каноничното представяне.

```
#include <iostream>   // Има недостатък
#include <cstring>
#include <cassert>
using namespace std;
class A
{ public:
      A(char* = "");
      ~A();
      A(const A&);
      A& operator=(const A &);
      void print() const
      { print_own();
      }
  protected:
      void print_own() const;
   private:
      char* x;
};
```

```
A::A(char* s)
{ x = new char[strlen(s)+1];
  assert(x != NULL);
  strcpy(x, s);
}

A::~A()
{ cout << "~A()\n";
  delete [] x;
}

A::A(const A& p)
{ x = new char[strlen(p.x)+1];
  assert(x != NULL);
  strcpy(x, p.x);
}
```

```
A& A::operator=(const A& p)
{ if (this != &p)
   { delete [] x;
     x = new char[strlen(p.x)+1];
     assert(x != NULL);
     strcpy(x, p.x);
   }
   return *this;
}

void A::print_own() const
{ cout << "A::x " << x << endl;
}
```

```
class B : virtual public A
{ public:
      B(char* = "", char* = "");
      ~B();
      B(const B&);
      B& operator=(const B&);
      void print() const
      { A::print_own();
        print_own();
      }
  protected:
      void print_own() const;
  private:
      char* x;
};
```

```
B::B(char* a, char* b) : A(a)
{ x = new char[strlen(b)+1];
  assert(x != NULL);
  strcpy(x, b);
}

B::~B()
{ cout << "~B()\n";
  delete [] x;
}

B::B(const B& p) : A(p)
{ x = new char[strlen(p.x)+1];
  assert(x != NULL);
  strcpy(x, p.x);
}
```

```
B& B::operator=(const B& p)
{ if (this != &p)
  { A::operator=(p);
    delete [] x;
     x = new char[strlen(p.x)+1];
     assert(x != NULL);
     strcpy(x, p.x);
  }
  return *this;
}

void B::print_own() const
{ cout << "B::x " << x << endl;
}
```

```
class C : virtual public A
{ public:
      C(char* = "", char* = "");
      ~C();
      C(const C&);
      C& operator=(const C&);
      void print() const
      { A::print_own();
        print_own();
      }
  protected:
      void print_own() const;
  private:
      char* x;
};
```

```
C::C(char* a, char* b) : A(a)
{ x = new char[strlen(b)+1];
  assert(x != NULL);
  strcpy(x, b);
}

C::~C()
{ cout << "~C()\n";
  delete [] x;
}

C::C(const C& p) : A(p)
{ x = new char[strlen(p.x)+1];
  assert(x != NULL);
  strcpy(x, p.x);
}
```

```
C& C::operator=(const C& p)
{ if (this != &p)
  { A::operator=(p);
    delete [] x;
    x = new char[strlen(p.x)+1];
    assert(x != NULL);
    strcpy(x, p.x);
  }
  return *this;
}

void C::print_own() const
{ cout << "C::x " << x << endl;
}
```

```
class D : public B, public C
{ public:
      D(char* = "", char* = "", char* = "",
         char* = "");
      ~D();
      D(const D&);
      D& operator=(const D&);
      void print() const
      { A::print_own(); B::print_own(); C::print_own();
        print_own();
      }
   protected:
      void print_own() const;
   private:
      char* x;
};
```

```
D::D(char* a, char* b, char* c, char* d) : A(a),
                                          B(a, b), C(a, c)
{ x = new char[strlen(d)+1];
  assert(x != NULL);
  strcpy(x, d);
}

D::~D()
{ cout << "~D() \n";
  delete [] x;
}

D::D(const D& p) : A(p), B(p), C(p)
{ x = new char[strlen(p.x)+1];
  assert(x != NULL);
  strcpy(x, p.x);
}
```

```
D& D::operator=(const D& p)
{ if (this!=&p)
   { B::operator=(p);
     C::operator=(p);
      delete [] x;
      x = new char[strlen(p.x)+1];
      assert(x != NULL);
      strcpy(x, p.x);
   }
   return *this;
}

void D::print_own() const
{ cout << "D::x " << x << endl;
}
```

```
int main()
{ D d("AAAA", "BBBB", "CCCC", "DDDD");
  d.print();
  D d1, d2;
  d2 = d1 = d;
  d1.print();
  d2.print();
  return 0;
}
```

В резултат три пъти се извежда
A::x AAAA
B::x BBBB
C::x CCCC
D::x DDDD

три пъти се извежда:
~D()
~C()
~B()
~A()