ПРЕОБРАЗУВАНЕ НА ТИПОВЕ – ОСОБЕНОСТИ. ШАБЛОНИ И НАСЛЕДЯВАНЕ. МНОЖЕСТВЕНО НАСЛЕДЯВАНЕ. ВИРТУАЛНИ КЛАСОВЕ И ФУНКЦИИ.

гл.ас., д-р. Нора Ангелова

Преобразуване на типове

ПРЕОБРАЗУВАНЕ НА ТИПОВЕ - ОСОБЕНОСТИ

- Ако наследяването на базовия клас е с атрибут public, възможно е заменяне на обекти.

Заменянето може да се извършва при:

1. инициализация;
2. присвояване;
3. предаване на параметри на функции;

Две посоки на замяна:

- „производен с основен“;
- „основен с производен“ ;

ПРЕОБРАЗУВАНЕ НА ТИПОВЕ - ОСОБЕНОСТИ

- „производен с основен“ - *неявни стандартни преобразувания.*

```
Der d;
Base b = d;

Base& b1 = d;

Der * d2 = &d;
Base * b2 =  d2;
```

ПРЕОБРАЗУВАНЕ НА ТИПОВЕ - ОСОБЕНОСТИ

- „производен с основен“ - *неявни стандартни преобразувания.*

```
Der d;
Base b = d;

Base& b1 = d;

Der * d2 = &d;
Base * b2 =  d2;
```

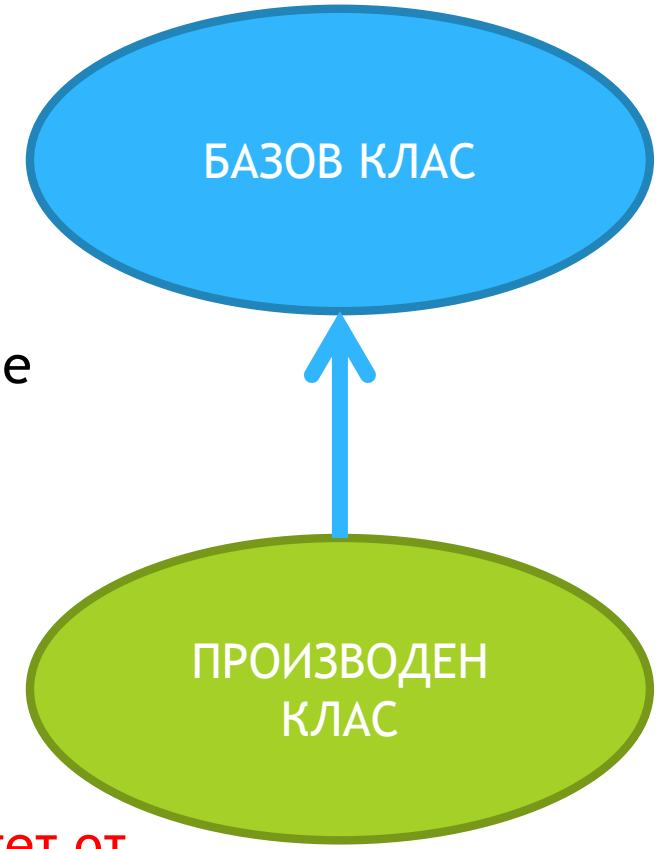
Достъп до собствените компоненти на производен клас:

1. чрез обект на основен клас (b) – не е възможно!
2. указател или псевдоним (b1, b2) – възможно с преобразувания

ПРЕОБРАЗУВАНЕ НА ТИПОВЕ - ОСОБЕНОСТИ

- „производен с основен“ - *неявни стандартни преобразувания.*

```
Der d;
Der * d2 = &d;
Base * b2 =  d2;

d2->derFunction(); // обичайно извикване
b2-> derFunction(); // недопустимо

((der*) b2)->derFunction(); // възможно
```



Забежка: ‘->’ и ‘.’ са с по-висок приоритет от преобразуването на типовете.

ПРЕОБРАЗУВАНЕ НА ТИПОВЕ - ОСОБЕНОСТИ

- „производен с основен“ - *неявни стандартни преобразувания.*

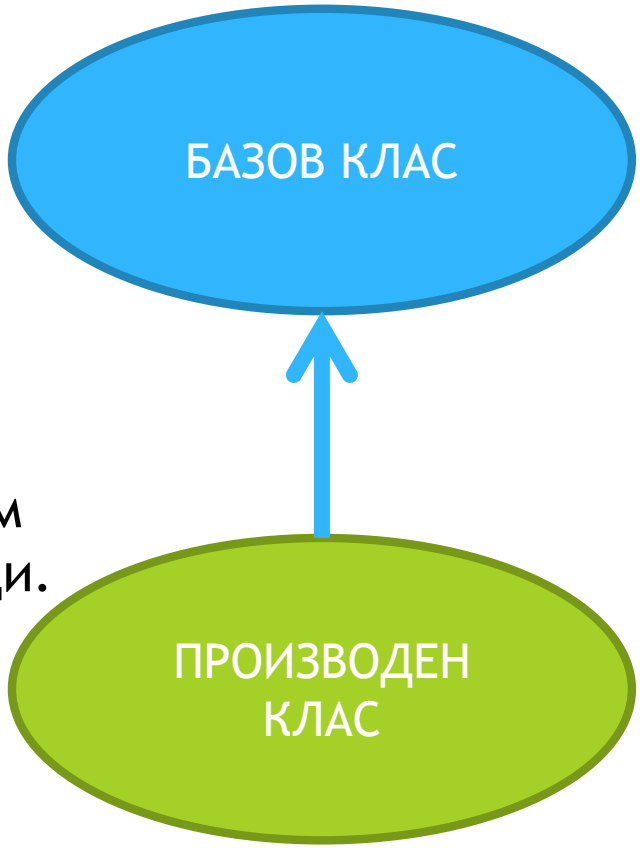
```
Der d;
Der & d2 = d;
Base & b2 =  d2;

d2.derFunction(); // обичайно извикване
b2. derFunction(); // недопустимо

((Der&) b2).derFunction(); // възможно
```

Забежка: ‘->’ и ‘.’ са с по-висок приоритет от преобразуването на типовете.

ПРЕОБРАЗУВАНЕ НА ТИПОВЕ - ОСОБЕНОСТИ

- „основен с производен“ – *явно преобразуване*

```
Base b;
Der d = (Der) b;
```



- Опасна операция!
- Собствените компоненти на обекта **d** останат неинициализирани.
  Опитът за използването (промяната) им може да доведе до сериозни последици.
- Съществуват реализации на езика, които няма да извършат това преобразуване.

ПРЕОБРАЗУВАНЕ НА ТИПОВЕ - ОСОБЕНОСТИ

- „основен с производен“ – *явно преобразуване*

```
Base b;
Der d = (Der) b;

Base * b1 = &b;
Der * d1 = (Der*) b1;

Der & d2 = (Der&) b;
```

- Указателят d1 не сочи истински обект от клас der.
- Съществуват реализации на езика, които няма да извършат това преобразуване.
- Няма собствени компоненти на класа der.
- Опит за използването им може да достъпи до памет, която е проделена за други цели.

УКАЗАТЕЛ КЪМ МЕТОД

- Указател към метод

```
void (Base::*bPtr)() = Base::bFunc;
```

Възможно е:

```
void (Der::*dPtr)() = bPtr;
Der y;
(y.*dPtr)();
```


```
void (Base::*bPtr)();
bPtr = (void (Base::*)()) Der::dFunc;
```

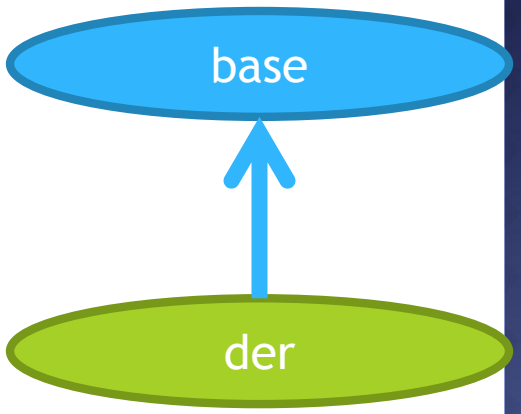
Използването може да доведе до грешка или неоднозначност.

Шаблони на класове и наследяване

Шаблони на класове и наследяване

- Шаблон на производен клас

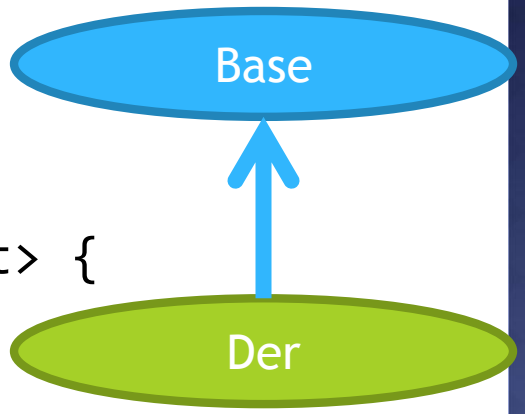
```
class Base { … };

template <class T>
class Der : <атрибут_за_област> Base {
 …
};
```



Шаблони на класове и наследяване

- Шаблон на базов клас

```
template <class T>
class Base { … };

class Der : <атрибут_за_област> Base<int> {
  …
};
// int или друг конкретен тип
```

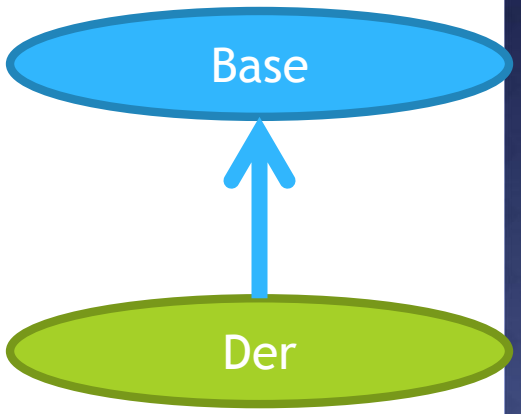


Шаблони на класове и наследяване

- Шаблон на базов и производен клас

Вариант 1

За всеки възможен базов клас съответства точно 1 производен клас.



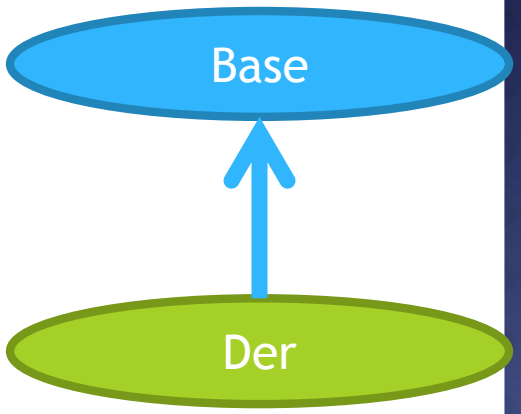

```
template <class T>
class Base { … };

template <class T>
class Der : <атрибут_за_област> Base<T> {
  …
};
```

Шаблони на класове и наследяване

- Шаблон на базов и производен клас

Вариант 2

За всеки възможен базов клас съответства множество от производни класове.




```
template <class T>
class Base { … };

template <class T, class U, …>
class Der : <атрибут_за_област> Base<T> {
  …
};
```

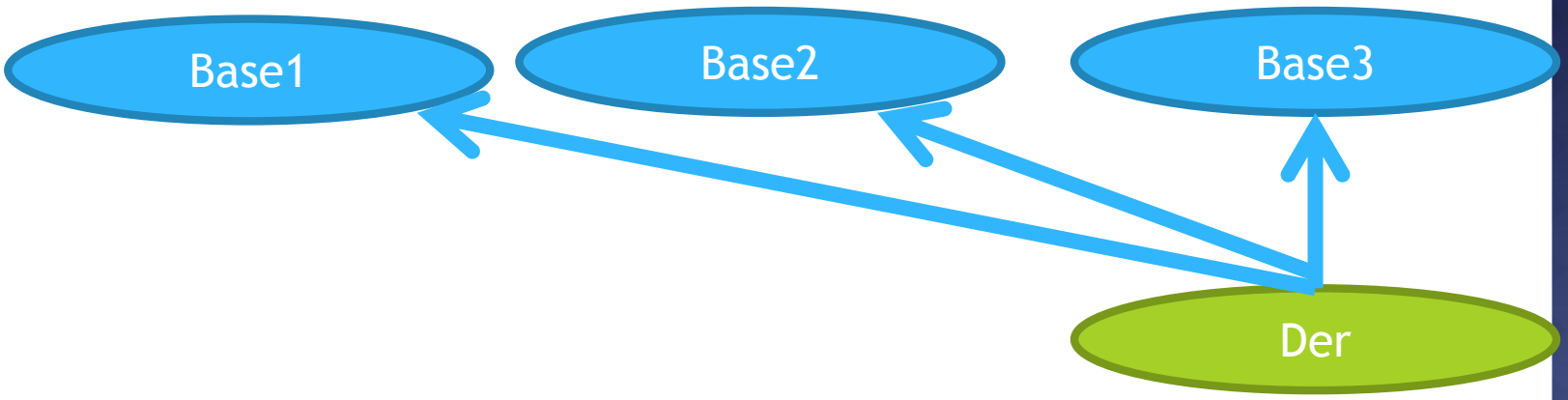
Множествено наследяване

Множествено наследяване

- Декларация на производен клас?

Можествено наследяване

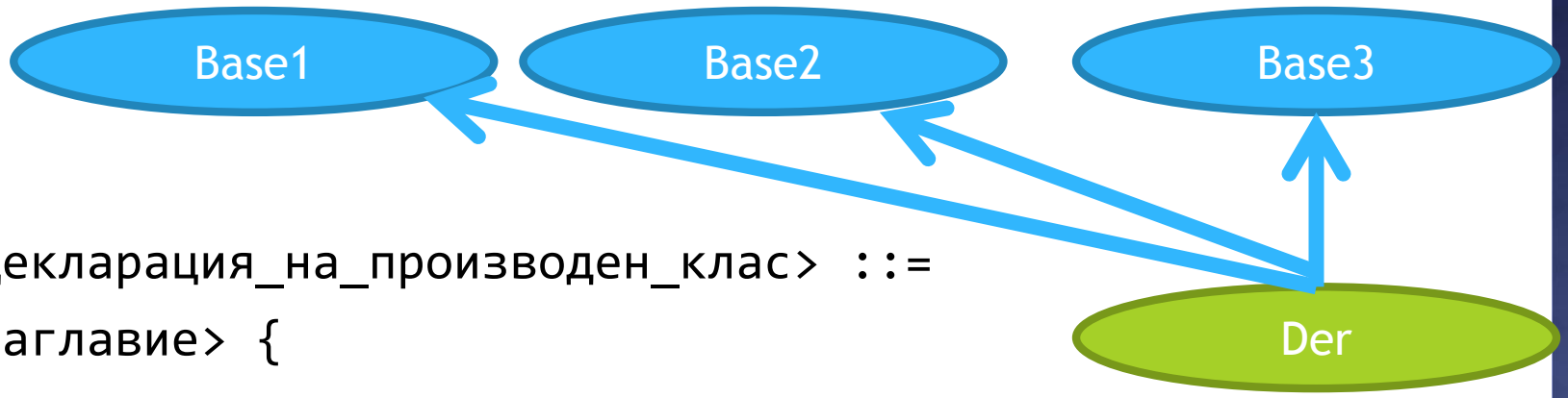
- Декларация на производен клас




```
<декларация_на_производен_клас> ::=
<заглавие> {
    <тяло>
};
```

МОЖЕСТВЕНО НАСЛЕДЯВАНЕ

- Декларация на производен клас




```
<декларация_на_производен_клас> ::=
<заглавие> {
   <тяло>
};
```

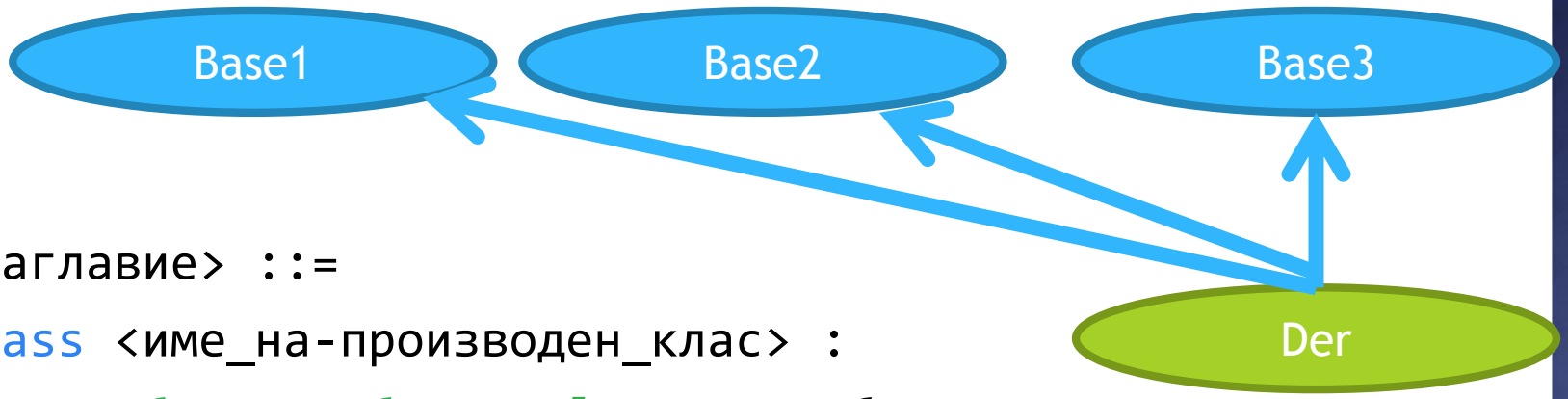
```
<заглавие> ::=
class <име_на-производен_клас> :
[ <атрибут_за_област> ] <име_на_базов_клас>,
[ <атрибут_за_област> ] <име_на_базов_клас>
{ , [ <атрибут_за_област> ] <име_на_базов_клас> }опц
```

Множествено наследяване

- Декларация на производен клас?
- Какви стойности може да приема атрибутът за област и коя е стойността по подразбиране?

Можествено наследяване

- Декларация на производен клас




```
<заглавие> ::=
class <име_на-производен_клас> :
[ <атрибут_за_област> ] <име_на_базов_клас>,
[ <атрибут_за_област> ] <име_на_базов_клас>
{ , [ <атрибут_за_област> ] <име_на_базов_клас> }опц

<атрибут_за_област> ::= public | protected | private
```

Множествено наследяване

- Декларация на производен клас?
- Какви стойности може да приема атрибутът за област и коя е стойността по подразбиране?
- Обяснете какви са атрибутите за област в дефиницията:

```
class Der : Base1, Base2, Base3
{ ...
};
```

Как си обяснявате това?

Множествено наследяване

- Производният клас наследява компонентите на всички базови класове като видът на наследяване се определя от атрибута за област на базовия клас.
- Правилата за наследяване, за пряк и външен достъп са същите като при единичното наследяване.

Множествено наследяване

- Декларация на производен клас?
- Какви стойности може да приема атрибутът за област и коя е стойността по подразбиране?
- Обяснете какви са атрибутите за област в дефиницията:

```
class Der : Base1, Base2, Base3 {
   ...
};
```

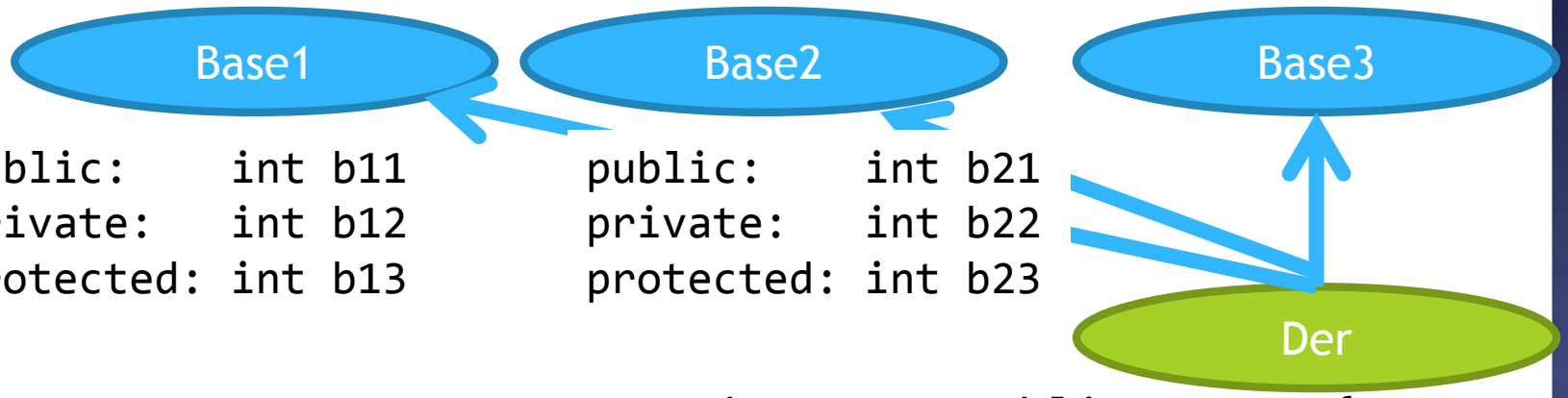
Как си обяснявате това?

- Какви са правилата за наследяване, за пряк и външен достъп при единичното наследяване.

МОЖЕСТВЕНО НАСЛЕДЯВАНЕ

- Декларация на производен клас

```
public:    int b31
private:   int b32
protected: int b33
```




```
public:    int b11
private:   int b12
protected: int b13
```

```
public:    int b21
private:   int b22
protected: int b23
```

```
class Der : Base1, protected Base2, public Base3 {
        public: …
        private: int d2;
        protected: …
};

der d;
```

Множествено наследяване

- За член-функциите на голямата четворка на производен клас с множествено наследяване са в сила аналогични правила, като при производен клас с единично наследяване. В общия случай тези член- функции на основните класове не се наследяват от производния им клас.

Множествено наследяване

- Декларация на производен клас?
- Какви стойности може да приема атрибутът за област и коя е стойността по подразбиране?
- Обяснете какви са атрибутите за област в дефиницията:

```
class Der : Base1, Base2, Base3
{ ...
};
```

Как си обяснявате това?

- Какви са правилата за наследяване, за пряк и външен достъп при единичното наследяване?
- Какви са правилата за член-функциите на голямата четворка на производен клас с единично наследяване?

Множествено наследяване

- Конструктор

При извикването на конструктор на производен клас последователно се изпълняват:

1) Конструкторите на базовите му класове в реда на тяхното задаване в декларацията на производния клас, а не в инициализиращия списък на конструктора.

   Ако за някой основен клас не е посочен конструктор в инициализиращия списък, изпълнява се конструкторът по подразбиране на класа, ако такъв е дефиниран (или може да се създаде), или се съобщава за грешка.
2) Конструкторите по подразбиране на класовете, чиито обекти са член-данни на производния клас, в случай, че в инициализиращият списък не е указано как да се инициализират. Редът на извикване съответства на реда на деклариране на тези член-данни в тялото на производния клас;
3) Тялото на конструктора на производния клас.

Множествено наследяване

Конструктор

- В някой от основните класове не е дефиниран конструктор в т.ч. за присвояване (такъв може да се генерира)

1. Ако в производния клас има дефиниран конструктор в инициализиращия списък не трябва да се прави обръщение към конструктор на основния клас му клас и наследената му част остава неинициализирана.

Множествено наследяване

Конструктор

- В някой от основните класове е дефиниран конструктор с параметри, от който не следва подразбиращият се конструктор

1. Ако в производния клас е дефиниран конструктор, в инициализиращия му списък задължително трябва да има обръщение към конструктора с параметъри на този основен клас.
2. Ако в производния клас не е дефиниран конструктор, компилаторът ще съобщи за грешка

Множествено наследяване

Конструктор

- В някой от основните класове са дефинирани няколко конструктора в т. ч. подразбиращ се

1. Ако в производния клас е дефиниран конструктор, в инициализиращия му списък може да не се посочи конструктор за този основен клас. Ще се използва подразбиращият се конструктор на основния клас.
2. Ако в производния клас не е дефиниран конструктор, компилаторът автоматично създава за него подразбиращ се конструктор.
   В този случай всички основни класове на производния клас трябва да имат конструктори по подразбиране

Множествено наследяване

- Деструктор

Всеки деструктор трябва да разруши само онези собствени компоненти, които са реализирани в динамичната памет

Извикването на деструкторите на базовите класове и производния им клас се осъществява автоматично в следната последователност:

1) извиква се деструкторът на производния клас,
2) в обратен ред, се извикват деструкторите на класовете на обектите, които са член-данни на производния клас (ако има такива),
3) изпълнявят се деструкторите на основните му класове, отново в обратен ред на реда на извикване на техните конструктори.

Множествено наследяване

- Конструктор за присвояване

Извиква конструктор за присвояване или друг подходящ, с който да се инициализират наследените член-данни

```
<инициализиращ_списък> ::=
<празно> |
: <име_на_основен_клас>(p)
{ , <име_на_основен_клас>(p) }
{ , <член-данна>(<параметри>) }
```

Множествено наследяване

Конструктор за присвояване

- в производния клас не е дефиниран конструктор за присвояване

Тогава компилаторът автоматично генерира за него конструктор за копиране, който преди да се изпълни активира и изпълнява конструкторите за присвояване (копиране) на всички основни класове в реда, указан в декларацията на производния клас.

В този случай конструкторите за присвояване (копиране) на основните класове се наследяват от производния клас.

Множествено наследяване

Конструктор за присвояване

- в производния клас е дефиниран конструктор за присвояване

Препоръчва се в инициализиращия му списък да има обръщения към конструкторите за присвояване на основните класове (ако такива са дефинирани).

Ако за някои основен клас не е указано такова обръщение, а е указан обикновен негов конструктор, инициализирането на наследените член-данни на този клас става чрез указания конструктор

Ако не е указано обръщение към конструктор за някой от основните класове, използва се конструкторът по подразбиране на основния клас, ако такъв съществува или се съобщава за отсъствието на подходящ конструктор за този основен клас, ако в него не е дефиниран конструктор по подразбиране.

Множествено наследяване

Операторна функция за присвояване

Присвояването на наследените член-данни става в тялото на операторната функция

```
<производен_клас>& <производен_клас>::operator=
(const <производен_клас>& p) {
 if (this != &p) {
  // Дефиниране на присвояването за наследените член-данни
  <основен_клас>::operator=(p);

  // Дефиниране на присвояването за собствените член-данни
  Del(); // разрушаване на онези собствени
  // член-данни на подразбиращия
  // се обект, които са разположени в ДП

  Copy(p); // копиране на собствените член-данни
  // на p в съответните член-данни на
  // подразбиращия се обект
 }
 return *this;
}
```

Множествено наследяване

Операторна функция за присвояване

- В производния клас не е дефинирана операторна функция за присвояване

Тогава компилаторът създава такава. Тя изпълнява операторните функции за присвояване (дефинирани или генерирани от компилатора) на всички основни класове на производния клас.

Множествено наследяване

Операторна функция за присвояване

- Ако в производния клас е дефинирана операторна функция за присвояване, тя трябва да се погрижи за присвояването на всички наследени член-данни.

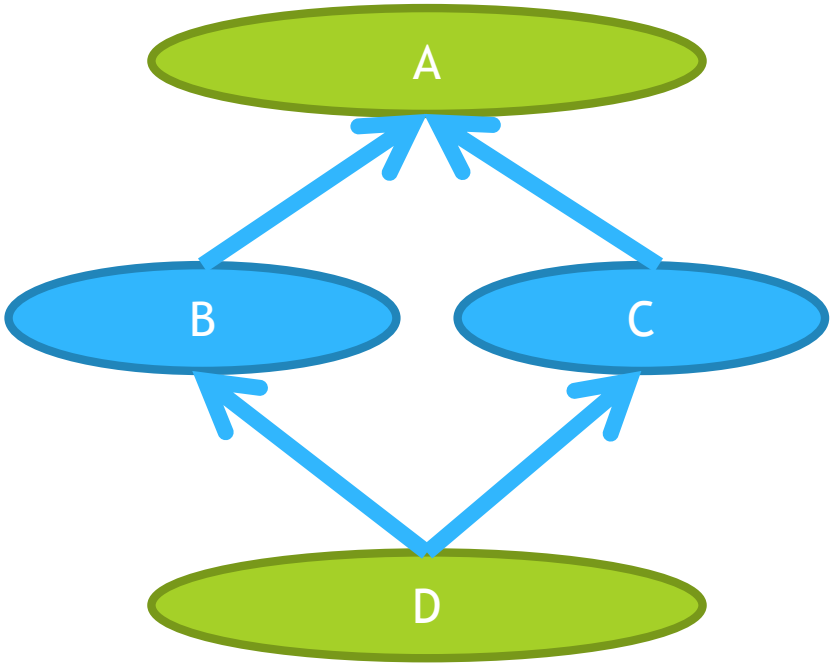
Ако това не е направено явно за някой основен клас, стандартът на езика не уточнява как ще стане присвояването на наследените от този клас член-данни.

ВИРТУАЛНИ КЛАСОВЕ

Виртуални класове

Проблем

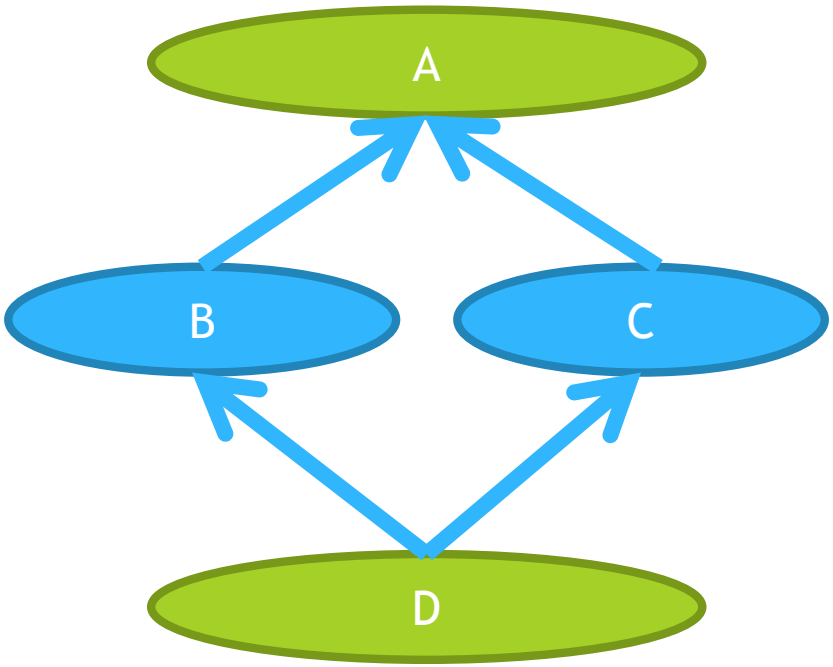
- Многократно наследяване на базов клас



ВИРТУАЛНИ КЛАСОВЕ

Проблем

- Многократно наследяване на базов клас – дублиране на член член-данните на класа в А, в класа Д



Клас D – собствени член-данни

клас B – собствени член-данни

клас B – наследени член-данни от клас A

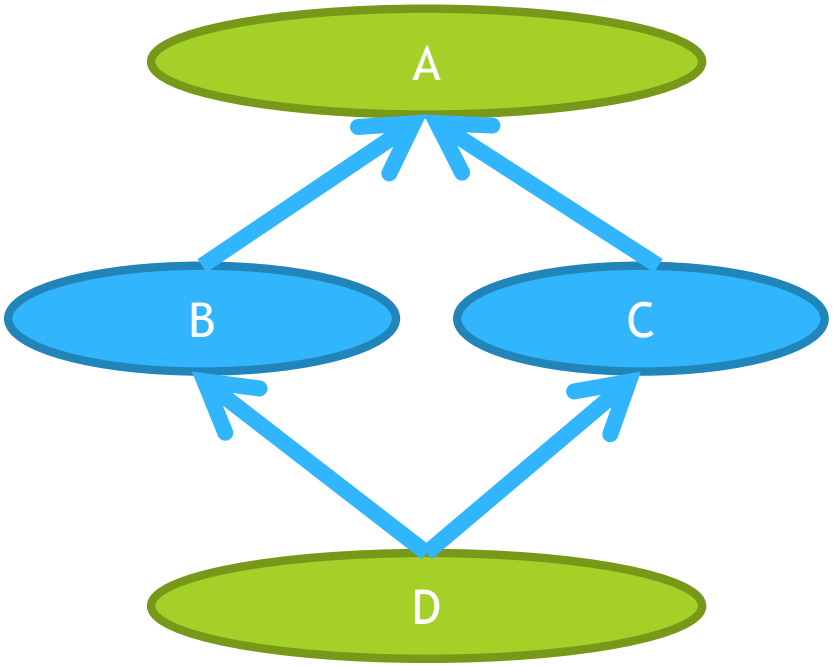
клас C – собствени член-данни

клас C – наследени член-данни от клас A

ВИРТУАЛНИ КЛАСОВЕ

Проблем

- Многократно наследяване на базов клас – дублиране на член член-данните на класа А, в класа Д



Клас D – собствени член-данни

клас B – собствени член-данни

клас B – наследени член-данни от клас A

клас C – собствени член-данни

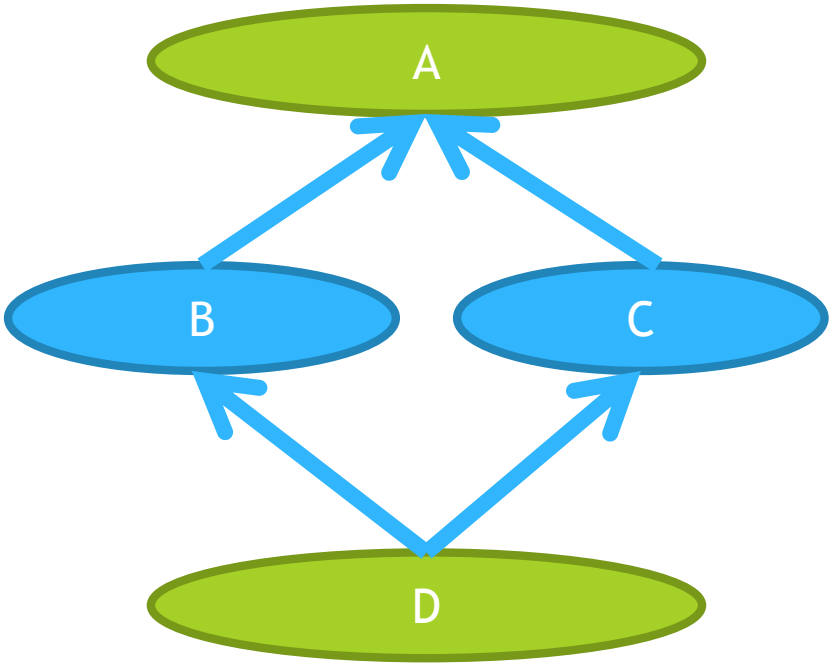
клас C – наследени член-данни от клас A

ВИРТУАЛНИ КЛАСОВЕ

Проблем

- Многократно наследяване на базов клас – неоднозначност на инициализация на член-данните.

```
// …
class C: public A {
  public:
  C(int a, int b) : A(a) { … }
};
// …
class B: public A {
  public:
  B(int a, int b) : A(a) { … }
};
// …
class D : public B, public C {
  public:
  D(int a, int b, int c, int d)
  : B(a, b), C(c, d)
  { }
};
```



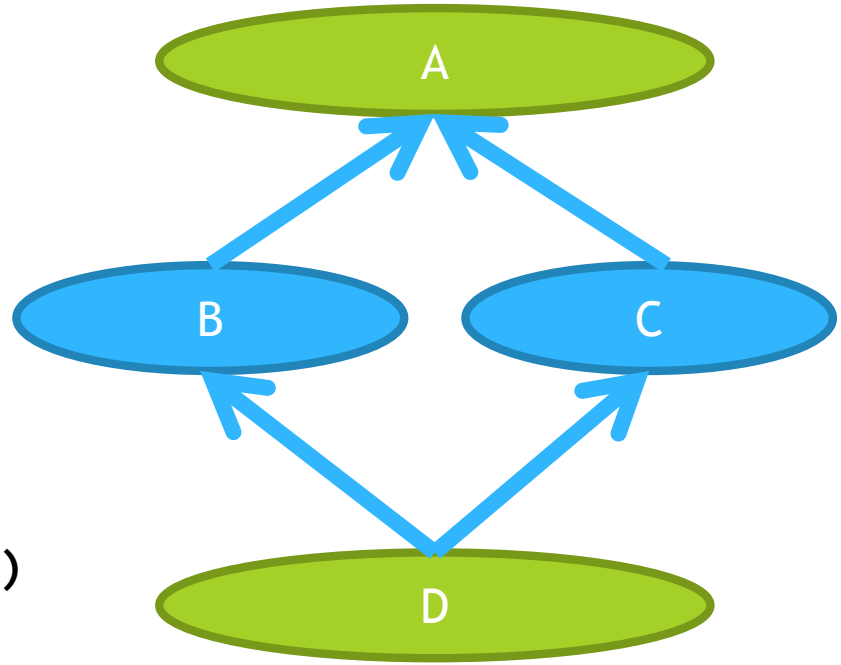

```
void main () { D(1, 2, 3, 4); } // D – член-данната на A е със стойност 1 или 3?
```

ВИРТУАЛНИ КЛАСОВЕ

Проблем

- Многократно наследяване на базов клас – неоднозначност при използване.

```
// …
class A {
 int x;
  public:
    A (int a) { x = a; }
    void print ();
};
// …
class D : public B, public C {
  public:
    D (int a, int b, int c, int d)
    : B(a, b), C(c, d) { }
    void func () {
      A::print(); // коя функция? this е от тип D *
    }
};
```



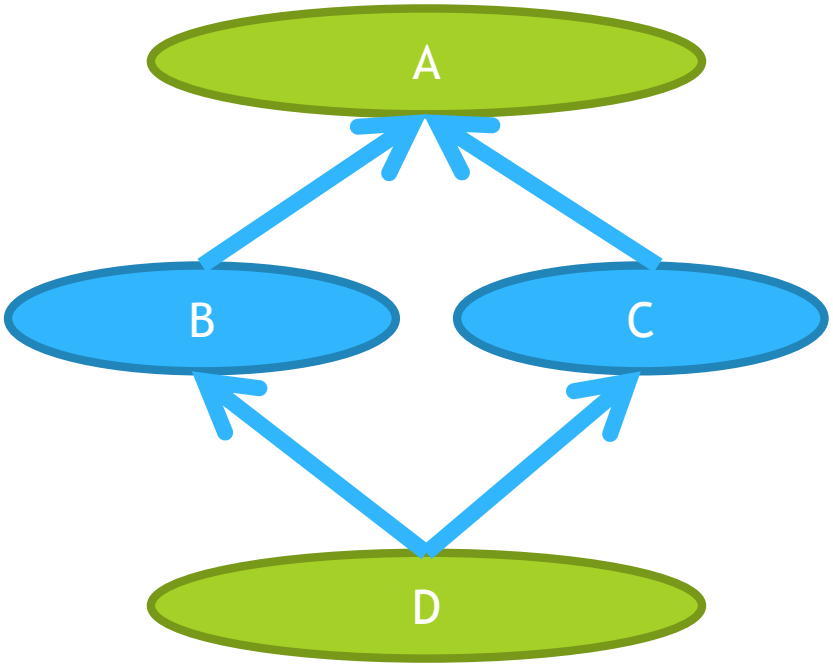
ВИРТУАЛНИ КЛАСОВЕ

Решение

- Многократно наследяване на базов клас – нееднозначност при използване.

Достъпът до „конфликтните“ компоненти на A става чрез последователно прилагане на операцията за явно преобразуване на типове. **Атрибутът трябва да е public.**

```
class A {
 int x;
  public:
    A (int a) { x = a; }
    void print ();
};
// …
class D : public B, public C {
  public:
    void func () {
      ((A)(C)*this).print();
      ((A)(B)*this).print();
    }
};

// …
D d(1, 2, 3, 4);
(A)(B) d;
(A)(C) d;
```



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

- Дават възможност за „поделяне“ на компонентите на основните класове.
- Създава се само едно тяхно копие.

Пример:

Класът D съдържа само едно копие на A.

- Декларира се като в декларацията на производния клас заедно с името и атрибута за област на основния клас се укаже и ключовата дума virtual.

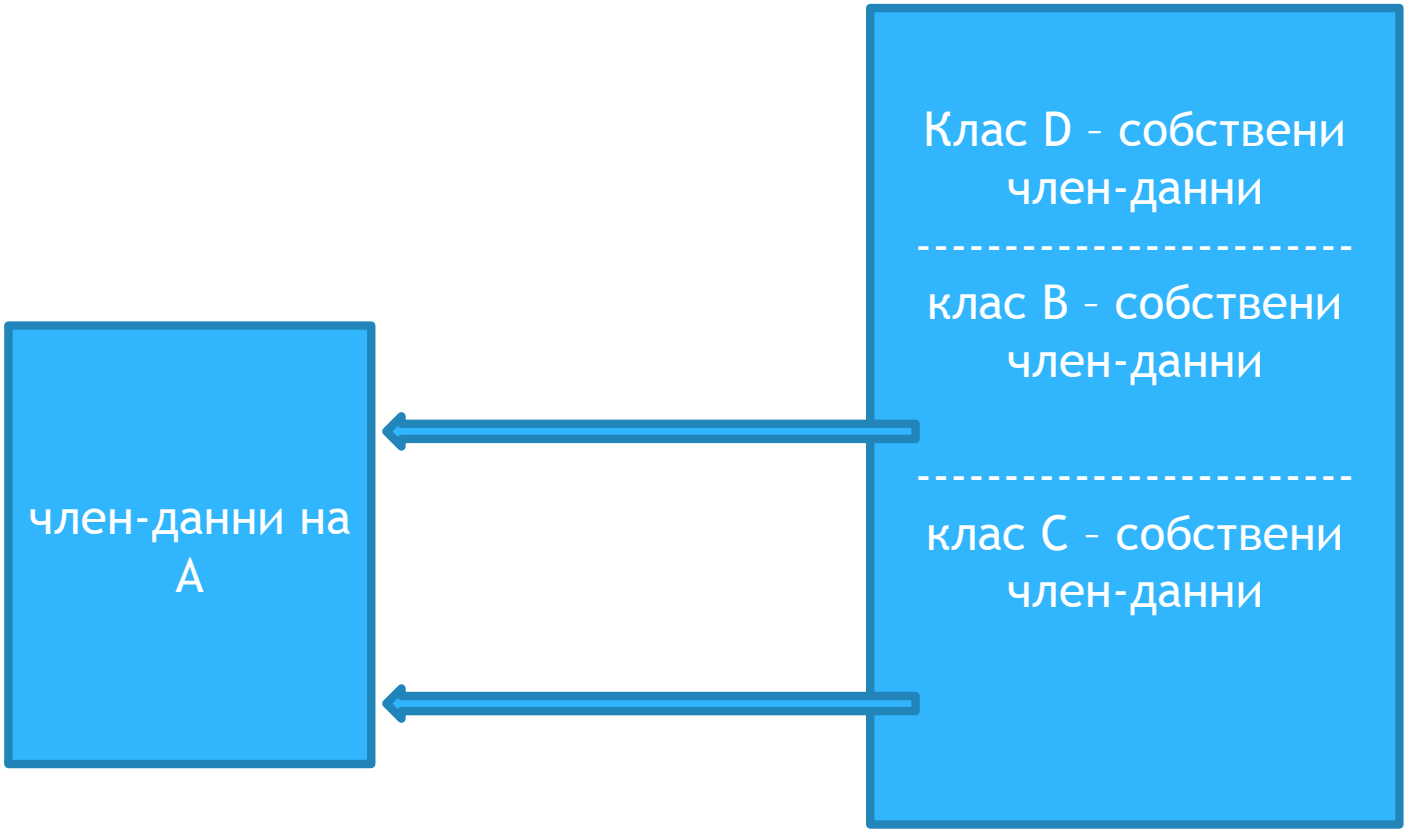
```
class Der : virtual public Base {};
```

- 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.

```
class D : public B, public C {
  public:
   D(int a, int b, int c, int d) : A(a), B(a,b), C(c,d) {}
    …
};
```

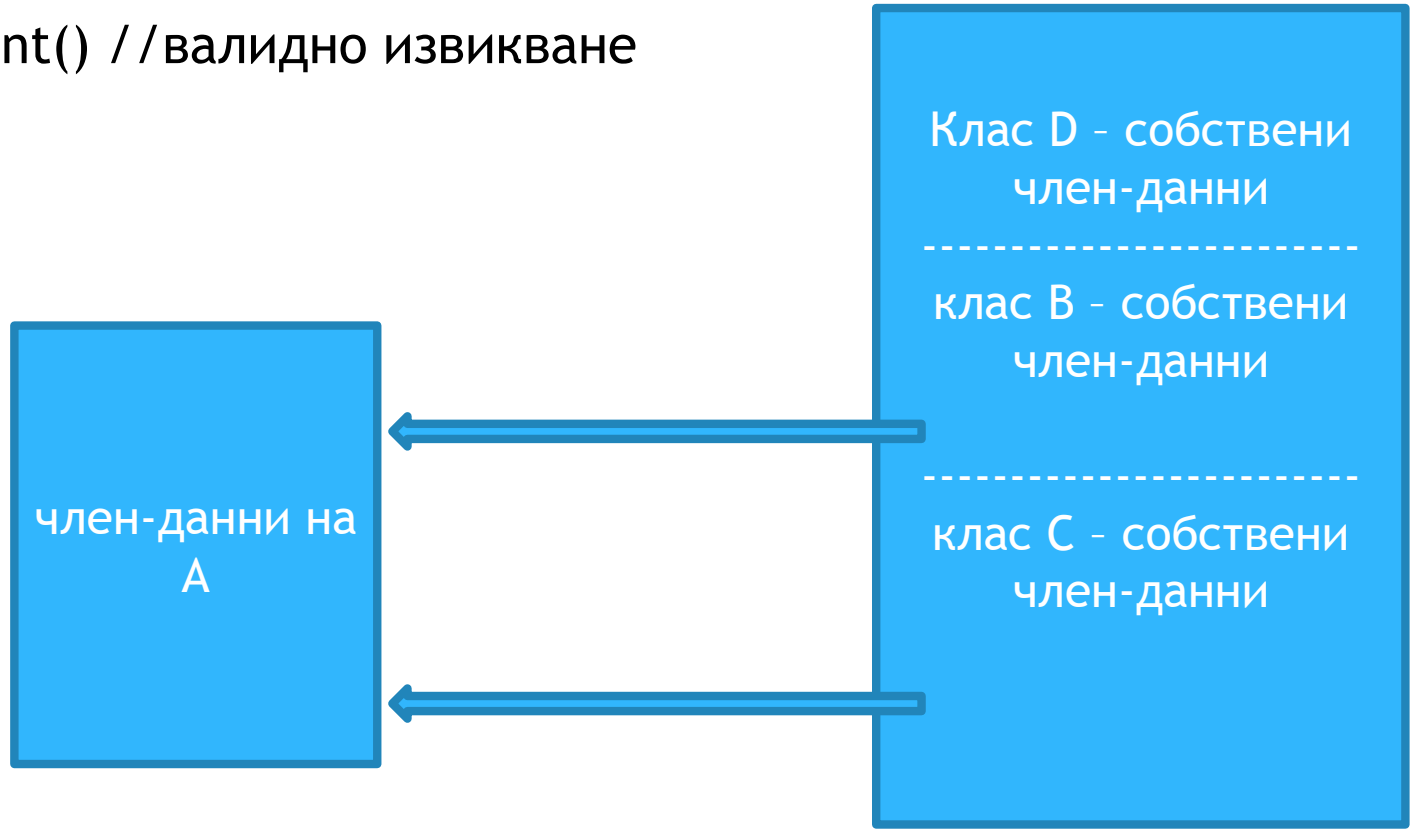
ВИРТУАЛНИ КЛАСОВЕ

- Инициализирането на виртуалните основни класове предхожда инициализирането на другите основни класове. (Първо се извикват техните конструктори).
- Редът на извикване на конструкторите става съгласно реда им в декларацията на производния клас.
- Конструкторът на виртуалния клас се извиква веднъж.



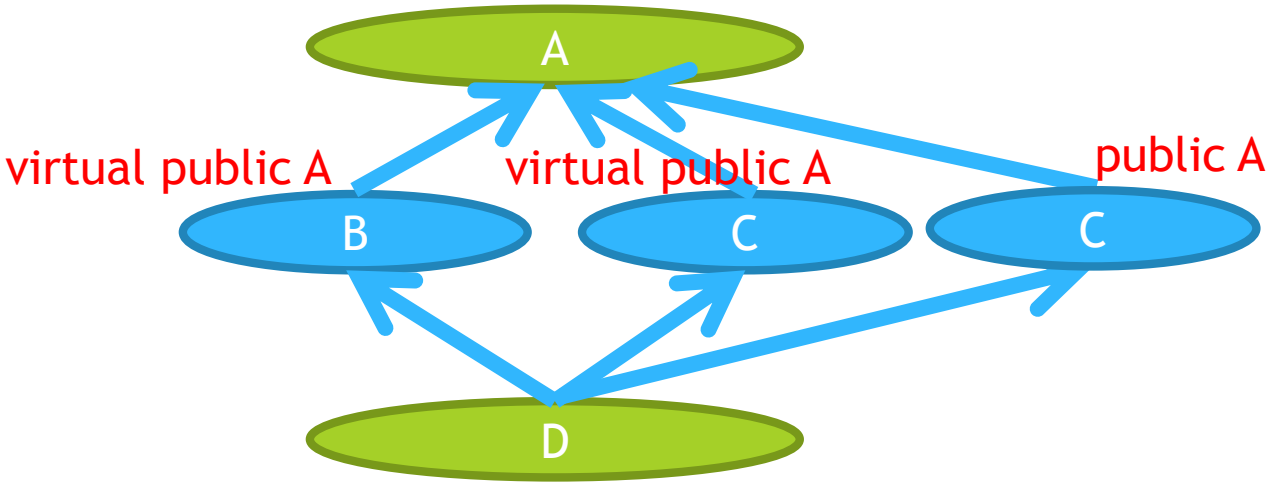
ВИРТУАЛНИ КЛАСОВЕ

- Инициализирането на виртуалните основни класове предхожда инициализирането на другите основни класове. (Първо се извикват техните конструктори).
- Редът на извикване на конструкторите става съгласно реда им в декларацията на производния клас.
- Конструкторът на виртуалния клас се извиква веднъж.
- A::print() //валидно извикване



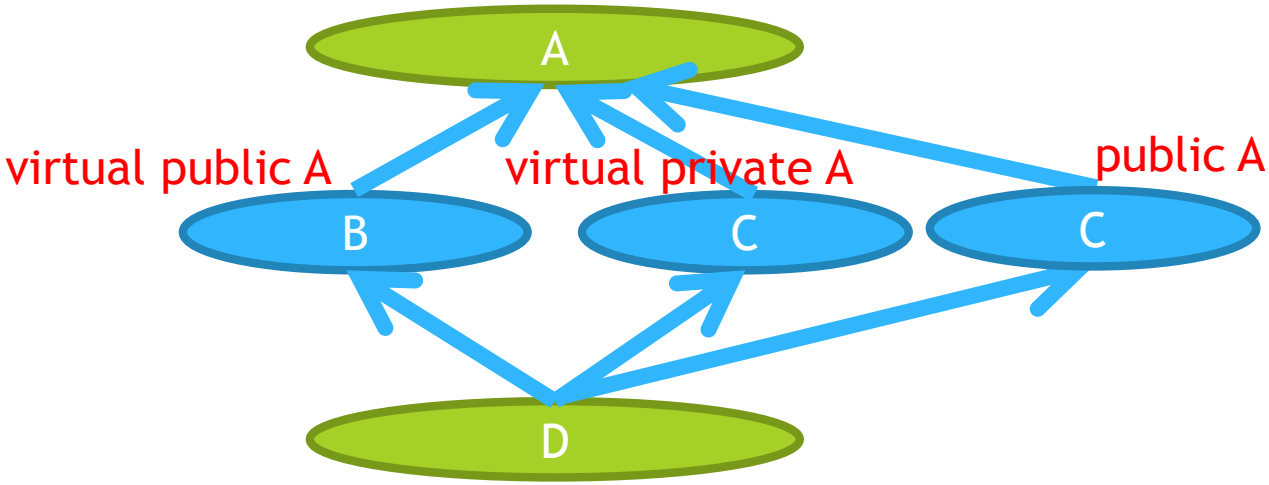
ВИРТУАЛНИ КЛАСОВЕ

- Инициализирането на виртуалните основни класове предхожда инициализирането на другите основни класове. (Първо се извикват техните конструктори).
- Редът на извикване на конструкторите става съгласно реда им в декларацията на производния клас.
- Конструкторът на виртуалния клас се извиква веднъж.
- A::print() //валидно извикване
- Възможно е смесено използване – трябва да се използва преобразуване.



ВИРТУАЛНИ КЛАСОВЕ

- Атрибути за област
  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.



Статично и динамично свързване

Статично и динамично свързване

- Статично свързване - изборът на функцията, която трябва да се изпълни става по време на компилация.

- Динамино свързване – изборът на функцията, която трябва да се изпълни става по време на изпълнение на програмата.

```
Point3 p3(1,2,3);
ColPoint3 p4(1,2,3,1);
Point2 * ptr1 = &p3;
ptr1->print(); // print на Point2

Point2 * ptr2 = &p4;
ptr2->print(); // print на Point2
```

Статично и динамично свързване

Динамино свързване

- Разширяването на йерархията не създава проблеми.
- Не се налага проверка на типа.
- Усложняване на кода и забавя процеса на изпълнение на програмата.
- Реализира се чрез специални член-функции на класове – виртуални член-функции.
- Виртуалните функции се декларират чрез поставяне на запазената дума virtual.

virtual <тип_на_резултата> <име_на_метод>(<параметри>);

Статично и динамично свързване

Динамино свързване

```
// Point2, Point3, ColPoint3
virtual void print();
```

```
Point3 p3(1,2,3);
ColPoint3 p4(1,2,3,1);
Point2 * ptr1 = &p3;
```

- Ще се определи по време на изпълнението на програмата
- Определянето е в зависимост от класа на обекта, към който сочи указателят, а не от класа, към който е указателят.

```
ptr1->print(); // print на Point3

Point2 * ptr2 = &p4;
ptr2->print(); // print на ColPoint3
```

Статично и динамично свързване

Динамино свързване

1. Само член-функциите на класовете могат да се декларират като виртуални.
2. Ако функция е обявена за виртуална в основния клас, декларираните член-функции в производните класове със същия прототип също са виртуални дори ако запазената дума бъде пропусната.
3. Ако в производен клас се дефинира виртуална функция, която има същия прототип като невиртуална функция в основния клас, двете функции се интерпретират като различни член-функции.
4. Възможно е виртуална функция да се дефинира извън клас. Тогава не започва със запазената дума virtual.
5. Виртуалните чрен-функции се наследяват като останалите компоненти на класа.
6. Основния клас, в който член-функция е обявена за виртуална, трябва да е с атрибут public в производните от него класове.
7. Виртуалните член-функциисе извикват чрез указател или псевдоним на обект на някакъв клас.
8. Виртуалната член-функция, която в действителност се изпълнява, зависи от класа на обекта, към който сочи указателят.

ПОЛИМОРФИЗЪМ

- Едни и същи действия се реализират по различен начин в зависимост от обектите, върху които се прилагат.
- Действията се наричат полиморфни.
- Свойство на член-функциите на класовете.
- Реализира се чрез виртуални функции.
- За да се реализира полиморфно действие, класовете върху, които ще се прилага, трябва да имат общ родител или прародител, т.е. да са производни на един и същ клас.
- В класа се дефинира виртуален метод, съответстващ на полиморфното действие.
- Всеки клас предефинира или не виртуалния метод.
- Активирането става чрез указател към базов клас, на който може да се присвоят адресите на обекти на който и да е от производните класове от йерархията.
- Ще се изпълни методът на съответния обект.

АБСТРАКТЕН КЛАС

- Ако класовете, в които трябва се дефинират виртуални методи, нямат общ родител, такъв може да бъде създаден изкуствено чрез т.нар. абстрактен клас.

- Клас, в който има поне една чисто виртуална функция.

virtual <тип_на_резултата> <име_на_метод>(<параметри>) = 0;

- Не могат да се създават обекти от тези класове, но могат да се дефинират указатели към такива класове.
- Чисто виртуалните функции задължително трябва да бъдат предефинирани в производните класове или да бъдат обявени като чисто виртуални в тях.

Виртуални деструктори

```
Base * b = new Der(1,2);

delete b; // какво се извиква, какво се разрушава?
```

КРАЙ