

**ВИРТУАЛНИ ФУНКЦИИ.
ПОЛИМОРФИЗЪМ.
ВИРТУАЛНИ ДЕСТРУКТОРИ.
АБСТРАКТНИ КЛАСОВЕ.**

гл.ас., д-р. Нора Ангелова

СТАТИЧНО И ДИНАМИЧНО СВЪРЗВАНЕ

СТАТИЧНО СВЪРЗВАНЕ

- Изборът на функцията, която трябва да се изпълни става по време на компилация.
- Изборът на функцията не може да се променя по време на изпълнение на програмата.

СТАТИЧНО СВЪРЗВАНЕ

Пример

Да се дефинира йерархия, определяща точка в равнината, точка в тримерното пространство и точка с цвят в тримерното пространство.

СТАТИЧНО СВЪРЗВАНЕ

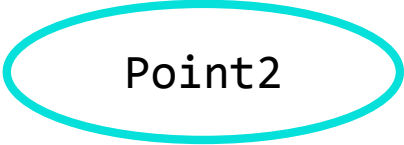
- Точка в равнината

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

```
class Point2 {
public:
    Point2(int x1 = 0, int y1 = 0) {
        x = x1;
        y = y1;
    }

    void print() const {
        cout << x << ", " << y;
    }

private:
    int x, y;
};
```

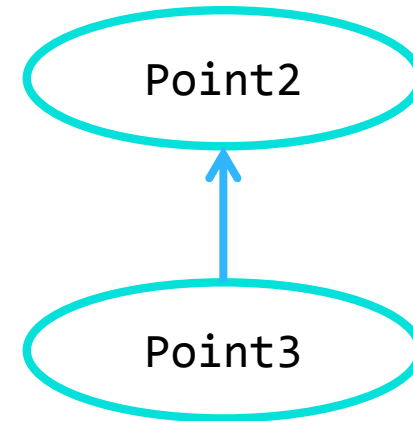


Point2

СТАТИЧНО СВЪРЗВАНЕ

Точка в тримерното пространство

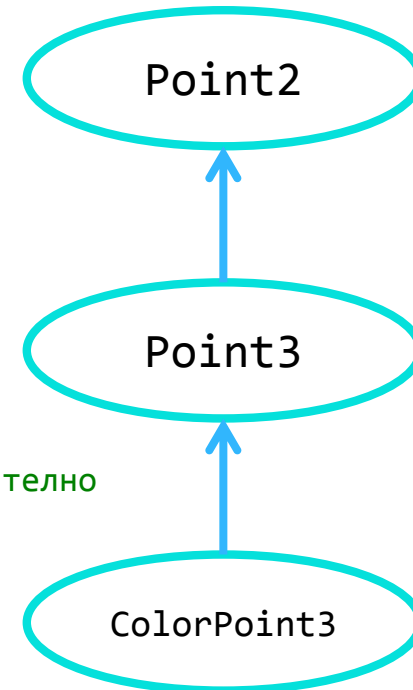
```
class Point3 : public Point2 {  
    public:  
        Point3(int x1 = 0, int y1 = 0, int z1 = 0) : Point2(x1, y1) {  
            z = z1;  
        }  
  
        void print() const {  
            Point2::print();  
            cout << ", " << z << endl;  
        }  
  
    private:  
        int z;  
};
```



СТАТИЧНО СВЪРЗВАНЕ

● Точка с цвят в тримерното пространство.

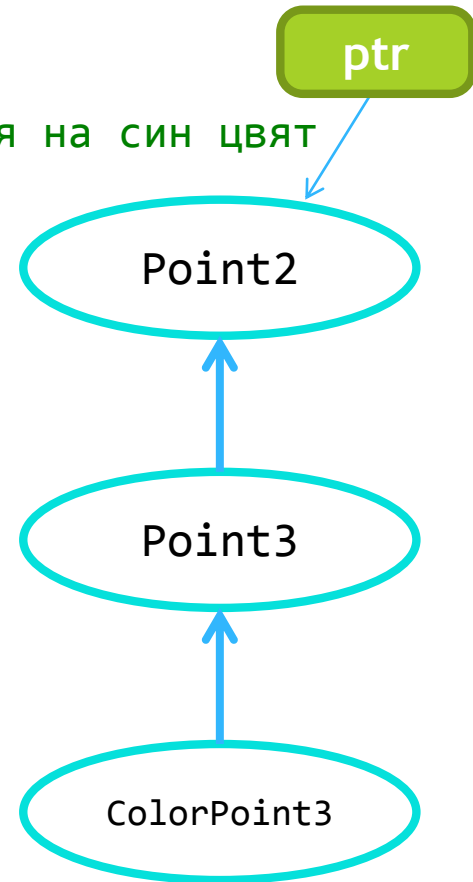
```
class ColorPoint3 : public Point3 {  
    public:  
        ColorPoint3(int x1 = 0, int y1 = 0, int z1 = 0, int c = 0) : Point3(x1, y1, z1) {  
            color = c;  
        }  
  
        void print() const {  
            Point3::print();  
            cout << "color: " << color << endl;  
        }  
  
    private:  
        int color; // число, което отговаря на цвят от предварително  
                  // дефинирана таблица с цветове  
};
```



СТАТИЧНО СВЪРЗВАНЕ

- Указатели към клас от йерархията

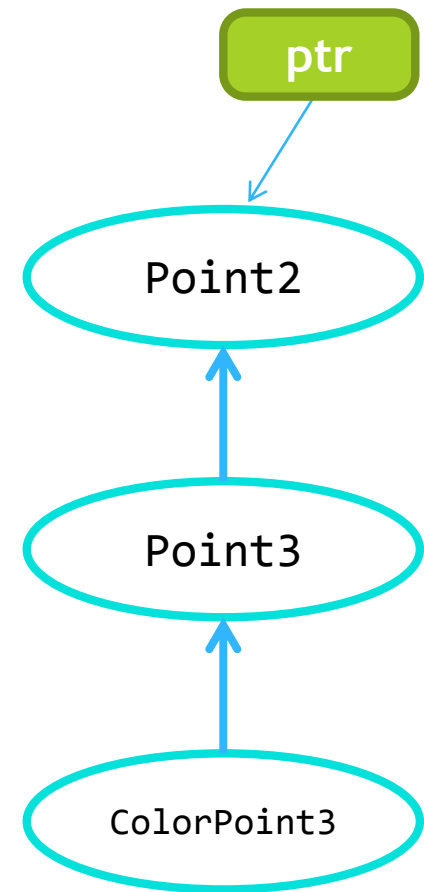
```
int main() {  
    Point2 p2(15, 10);  
    Point3 p3(21, 41, 63);  
    ColorPoint3 p4(12, 24, 36, 11); // 11 отговаря на син цвят  
  
    Point2 *ptr = &p2;  
    ptr->print();  
    cout << endl;  
  
    ptr = &p3; // атрибутът на Point2 е public  
    ptr->print();  
    cout << endl;  
  
    ptr = &p4; // атрибутът на Point3 е public  
    ptr->print();  
    cout << endl;  
  
    return 0;  
}
```



СТАТИЧНО СВЪРЗВАНЕ

- Указатели към клас от йерархията

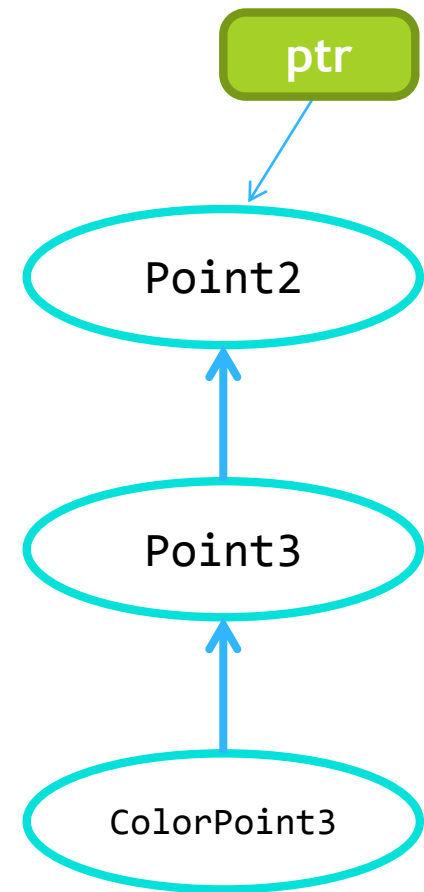
```
int main() {  
    Point2 p2(15, 10);  
    Point3 p3(21, 41, 63);  
    ColorPoint3 p4(12, 24, 36, 11);  
  
    Point2 *ptr = &p2;  
    ptr->print(); Point2::print() - 15, 10  
    cout << endl;  
  
    ptr = &p3; // атрибутът на Point2 е public  
    ptr->print(); Point2::print() - 21, 41  
    cout << endl;  
  
    ptr = &p4; // атрибутът на Point3 е public  
    ptr->print(); Point2::print() - 12, 24  
    cout << endl;  
  
    return 0;  
}
```



СТАТИЧНО СВЪРЗВАНЕ

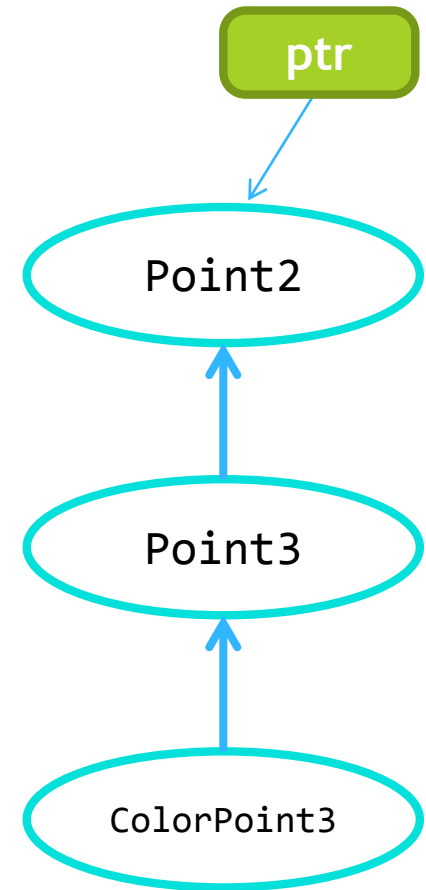
- Указатели към клас от йерархията

```
int main() {  
    Point2 p2(15, 10);  
    Point3 p3(21, 41, 63);  
    ColorPoint3 p4(12, 24, 36, 11);  
  
    Point2 *ptr = &p2;  
    ptr->print(); Point2::print() - 15, 10  
    cout << endl;  
  
    ptr = &p3; // атрибутът на Point2 е public  
    ((Point3*)ptr)->print(); Point3::print() - 21, 41, 63  
    cout << endl;  
  
    ptr = &p4; // атрибутът на Point3 е public  
    ((ColorPoint3*)ptr)->print(); ColorPoint3::print() - 12, 24, 36  
    cout << endl;                                color: 11  
  
    return 0;  
}
```



СТАТИЧНО СВЪРЗВАНЕ

- За да се извикат правилните функции трябва да се предвидят възможните обекти, указатели и псевдоними на обекти, чрез които ще се извикват член-функциите.
- Какво ще стане при по-сложни йерархии ?!?



ДИНАМИЧНО СВЪРЗВАНЕ

- Изборът на функцията, която трябва да се изпълни става по време на изпълнение на програмата.
- Не се налага явно преобразуване на типове.
- Опростяват се текстовете на програмите.
- Разширяването на йерархията не е проблем.
- Усложняване на кода и забавяне на програмата.
- Реализира се с виртуални член-функции.

ДИНАМИЧНО СВЪРЗВАНЕ

- Виртуални функции

`virtual` [<тип_на_резултата>] <име_на_метод> (<параметри>) [`const`];

думата `virtual` се поставя пред декларацията на функцията

ДИНАМИЧНО СВЪРЗВАНЕ

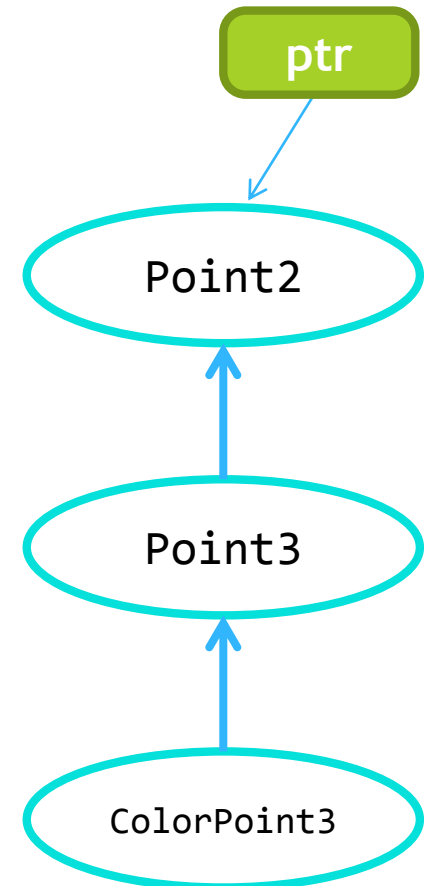
Виртуални функции

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

```
class Point2 {
public:
    Point2(int x1 = 0, int y1 = 0) {
        x = x1;
        y = y1;
    }

    virtual void print() const {
        cout << x << ", " << y;
    }

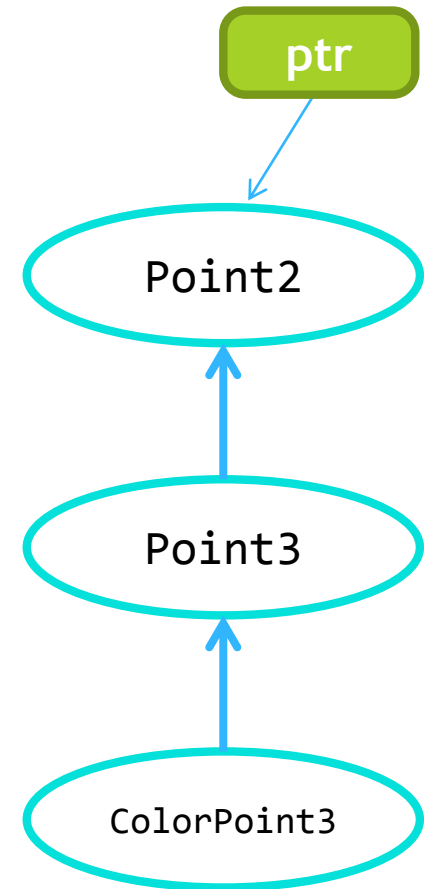
private:
    int x, y;
};
```



ДИНАМИЧНО СВЪРЗВАНЕ

Виртуални функции

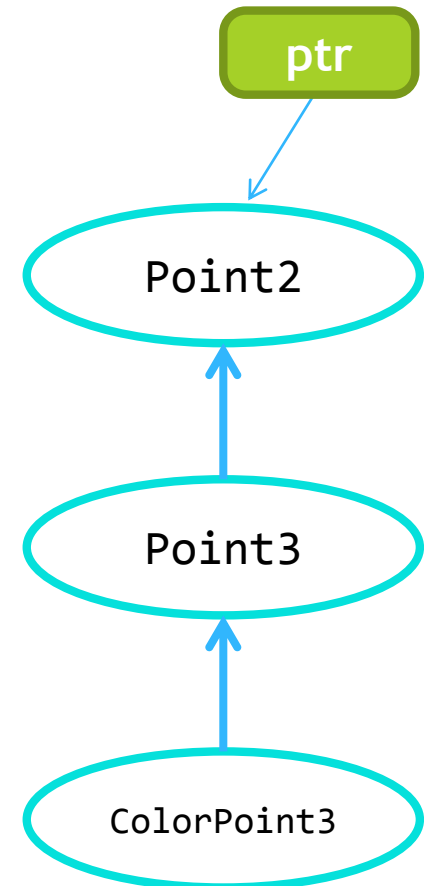
```
class Point3 : public Point2 {  
    public:  
        Point3(int x1 = 0, int y1 = 0, int z1 = 0) : Point2(x1, y1) {  
            z = z1;  
        }  
  
        virtual void print() const {  
            Point2::print();  
            cout << ", " << z << endl;  
        }  
  
    private:  
        int z;  
};
```



ДИНАМИЧНО СВЪРЗВАНЕ

Виртуални функции

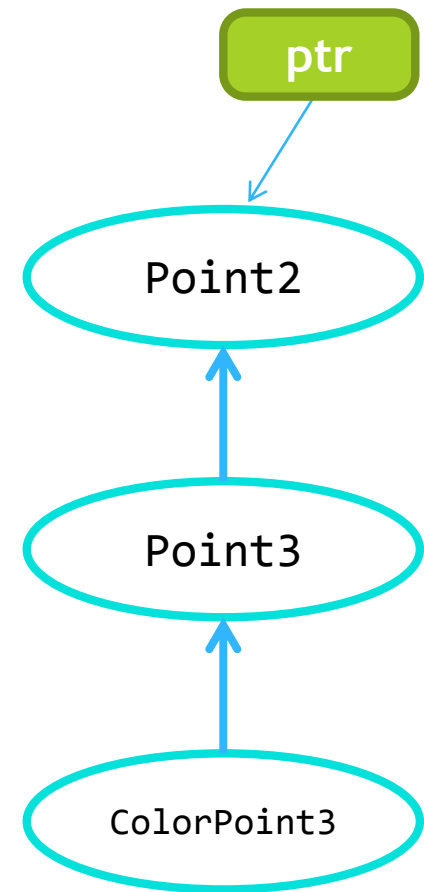
```
class ColorPoint3 : public Point3 {  
    public:  
        ColorPoint3(int x1 = 0, int y1 = 0, int z1 = 0, int c = 0) : Point3(x1, y1, z1) {  
            color = c;  
        }  
  
        virtual void print() const {  
            Point3::print();  
            cout << "color: " << color << endl;  
        }  
  
    private:  
        int color; // число, което отговаря на цвят от предварително  
                  // дефинирана таблица с цветове  
};
```



СТАТИЧНО СВЪРЗВАНЕ

- Указатели към клас от йерархията

```
int main() {  
    Point2 p2(15, 10);  
    Point3 p3(21, 41, 63);  
    ColorPoint3 p4(12, 24, 36, 11);  
  
    Point2 *ptr = &p2;  
    ptr->print(); Point2::print() - 15, 10  
    cout << endl;  
  
    ptr = &p3; // атрибутът на Point2 е public  
    ptr->print(); Point3::print() - 21, 41, 63  
    cout << endl;  
  
    ptr = &p4; // атрибутът на Point3 е public  
    ptr->print(); ColorPoint3::print() - 12, 24, 36  
    cout << endl; color: 11  
  
    return 0;  
}
```



ДИНАМИЧНО СВЪРЗВАНЕ

- Декларирането на член -функциите print като виртуални причинява трите обръщания към print чрез указателя ptr да определят функцията, която ще бъде извикана по време на изпълнението на програмата.
- Определянето е в зависимост от типа на обекта, към който сочи указателят, а не от класа, от който е указателят.

ДИНАМИЧНО СВЪРЗВАНЕ

- В случая `ptr = &p3`, указателят `ptr` е от класа `Point2`, но сочи обекта `p3`, който е от класа `Point3`.
Затова обръщението `ptr->print()` ще изпълни `Point3::print()`, ако е възможен достъп.
- В случая `ptr = &p4`, указателят `ptr` сочи обекта `p4`, който е от класа `ColorPoint3`. Затова обръщението `ptr->print()` ще изпълни `ColorPoint3::print()`, ако е възможен достъп.

И в двата случая достъпът е възможен, тъй като `ptr` е от тип `Point2*`, а в този клас член-функцията `print` е в `public` секцията му.

ДИНАМИЧНО СВЪРЗВАНЕ

◎ Свойства

1. Само член-функциите на класовете могат да се декларират като виртуални. Конструкторите не 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 неvirtуална 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 член-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-функции се 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.
9. Локалният и външният достъпът до виртуална член-функция имат някои особености.

ДОСТЪП ДО ВИРТУАЛНА ЧЛЕН-ФУНКЦИИ

- Пряк достъп (вътрешен, локален)

Виртуална член-функция на производен клас има пряк достъп до:

- собствените на производния клас компоненти
- компонентите, декларирани в `public` и `protected` секциите на основните си класове.

Всяка член-функция на клас, в който е дефинирана виртуална член-функция, има пряк достъп както до виртуалната член-функция на самия клас, така и до виртуалните член-функции със същия прототип на производните му директни и индиректни класове без значение на вида на секциите, в които са дефинирани виртуалните член-функции.

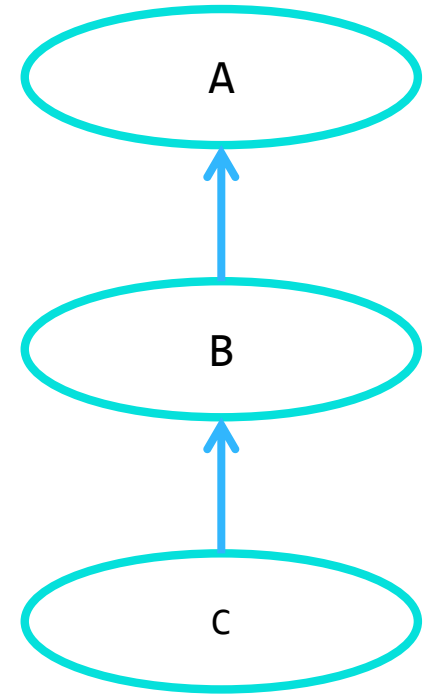
ДОСТЪП ДО ВИРТУАЛНА ЧЛЕН-ФУНКЦИИ

- ◎ Прям достъп (вътрешен, локален)

```
class A {  
    private:  
        virtual void f() const {  
            cout << "A\n";  
        }  
    public:  
        void test() const {  
            f(); // разрешава се динамично  
        }  
};
```

```
class B : public A {  
    protected:  
        void f() const {  
            cout << "B\n";  
        }  
};
```

```
class C : public B {  
    private:  
        virtual void f() const {  
            cout << "C\n";  
        }  
};
```



Член-функцията test на класа A има пряк достъп до:

- виртуалната член-функция f на класа A,
- виртуалните член-функции f на класовете B и C

ДОСТЪП ДО ВИРТУАЛНА ЧЛЕН-ФУНКЦИИ

- ◎ Прям достъп (вътрешен, локален)

```
class A {  
    private:  
        virtual void f() const {  
            cout << "A\n";  
        }  
    public:  
        void test() const {  
            f(); // разрешава се динамично  
        }  
};
```

```
class B : public A {  
    protected:  
        void f() const {  
            cout << "B\n";  
        }  
};
```

```
class C : public B {  
    private:  
        virtual void f() const {  
            cout << "C\n";  
        }  
};
```

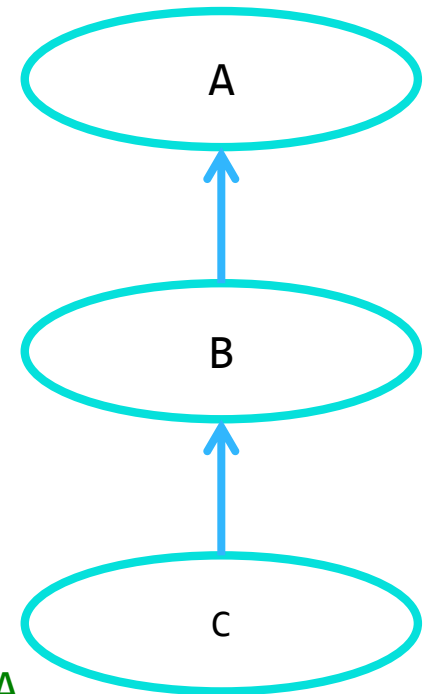
```
int main() {  
    A a; B b; C c;  
    A *ptr = &a;  
    ptr->test(); // A
```

```
    ptr = &b;  
    ptr->test(); // B
```

```
    ptr = &c;  
    ptr->test(); // C
```

```
    return 0;
```

```
}
```



ДОСТЪП ДО ВИРТУАЛНА ЧЛЕН-ФУНКЦИИ

◉ Външен достъп

1. Външният достъп до виртуална член-функция на клас чрез обект на класа се определя по традиционните правила. Връзката се разрешава статично.
2. Външен достъп чрез указател

```
клас1* ptr = & обект;  
ptr -> виртуална_функция_на_клас1(...);
```

Ако *виртуална_функция_на_клас1* е в **public** секция в *клас1*.

3. Външен достъп чрез псевдоним

```
клас1& ptr = обект;  
ptr . виртуална_функция_на_клас1(...);
```


ПОЛИМОРФИЗЪМ

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.

ПОЛИМОРФИЗЪМ

- 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.

ПОЛИМОРФИЗЪМ

- Ако класовете, над които ще се реализира полиморфно действие, нямат общ родител, такъв може да бъде създаден изкуствено чрез дефиниране на т.н. абстрактен клас.

ПОЛИМОРФИЗЪМ

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

Да разгледаме пример с класове от животни:

- Класове за котка и куче;
- Метод за извеждане на звук, който издава съответното животно;
- Извикване на методите ?

Друг пример за полиморфно действие:

- Класове с член-функции с еднакви прототипи;
- Член-функциите извършват еднотипни действия;
- Член-функциите на производните класове обикновено извършват редица общи действия.
- Метод за извеждане на името на зоологическата градина и животното.
- В този случай в основния клас може да се реализира една неvirtуална функция, която извършва общите действия и след (или преди) това извиква виртуалната функция, извършваща специфичните действия за класовете.

ПОЛИМОРФИЗЪМ

```
class ZooAnimal {
public:
    void print() const {
        cout << "ZooAnimal\n";
        cout << "Address:\n Sofia, Bulgaria\n";
    }
};

class Cat : public ZooAnimal {
public:
    void print() const {
        cout << "ZooAnimal\n" << "Cat\n";
    }
};

class Mouse : public ZooAnimal {
public:
    void print() const {
        cout << "ZooAnimal\n" << "Mouse\n";
    }
};

class Rabbit : public ZooAnimal {
public:
    void print() const {
        cout << "ZooAnimal\n" << "Rabbit \n";
    }
};
```



```
class ZooAnimal {
public:
    virtual void spec() const {
        cout << "Address:\nSofia, Bulgaria\n";
    }

    void print() const {
        cout << "ZooAnimal\n";
        spec(); // разрешава се динамично
    }
};

class Cat : public ZooAnimal {
public:
    virtual void spec() const {
        cout << "Cat\n";
    }
};

class Mouse : public ZooAnimal {
public:
    virtual void spec() const {
        cout << "Mouse\n";
    }
};

class Rabbit : public ZooAnimal {
public:
    virtual void spec() const {
        cout << "Rabbit\n";
    }
};
```

ПОЛИМОРФИЗЪМ

Съществуват три случая, при които обръщение към виртуална член-функция се разрешава статично:

- Виртуалната функция се извиква чрез обект на класа, в който е дефинирана.

```
Cat myCat; myCat.spec();
```

- Виртуалната член-функция се активира чрез указател към или чрез псевдоним на обект, но явно, чрез оператора ::, е посочена конкретната виртуална член-функция.

```
ZooAnimal *animalPtr = &myCat;  
animalPtr -> spec(); // динамично свързване  
animalPtr->ZooAnimal::spec(); // статично свързване
```

- Виртуалната член-функция се активира в тялото на конструктор или деструктор на основен клас.

В този случай се изпълнява виртуалната член-функция на основния клас. Това е така, защото виртуалната функция в конструктора или деструктора на основния клас се извиква когато обектът от производния клас още не е създаден или вече е разрушен.

ВИРТУАЛНИ ДЕСТРУКТОРИ

ВИРТУАЛНИ ДЕСТРУКТОРИ

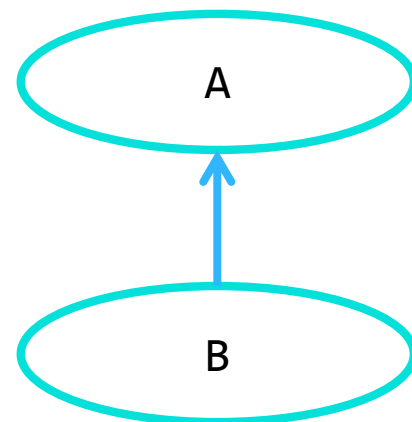
```
A * ptr = new B("1234", "stringB");
```

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

- Извиква се конструктор на A;
- Извиква се конструктор на B;
- Извиква се деструктор на A;
ptr е от тип A*.
Обръщението `delete ptr;`
ще изпълни деструктора на A.

stringB не се разрушава

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



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

ВИРТУАЛНИ ДЕСТРУКТОРИ

```
A * ptr = new B("1234", "stringB");
```

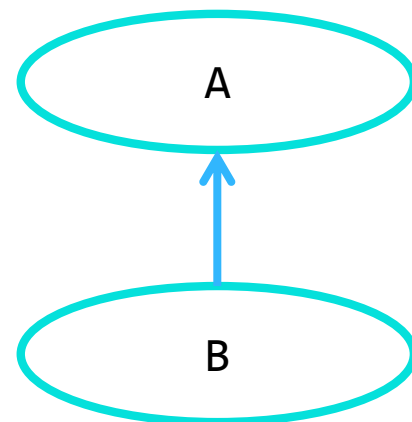
```
delete ptr; // какво се извиква, какво се разрушава?
```

Решение:

- Деструкторът на базовия клас A на йерархията се обяви за виртуален.

Обявяването на деструктор на клас на йерархия за виртуален причинява деструкторите на всички класове в наследствената за този основен клас йерархия да са виртуални.

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



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

ВИРТУАЛНИ ДЕСТРУКТОРИ

```
A * ptr = new B("1234", "stringB");
```

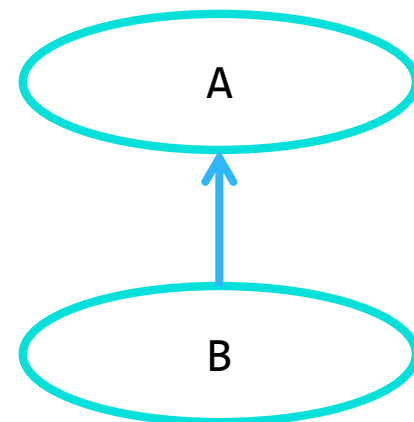
```
delete ptr; // какво се извиква, какво се разрушава?
```

Решение:

- Обявяваме деструктора на класа A за виртуален.

```
virtual A::~A() {  
    delete [] str;  
}
```
- Деструкторът на класа B автоматично става виртуален.
- `delete ptr;` ще изпълни:
 - деструктора на класа B,
 - деструктора на класа A,
 - ще разруши връзката на ptr с ДП.

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



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

АБСТРАКТНИ КЛАСОВЕ

ЧИСТО ВИРТУАЛНА ФУНКЦИЯ

- Възможно е виртуална член-функция да не е дефинирана, а само декларирана в клас. Такава виртуална член- функция се нарича чисто виртуална.

```
virtual [<тип_на_резултата>] <име_на_метод> (<параметри>) [const] = 0;
```

АБСТРАКТЕН КЛАС

- Клас, в който има поне една чисто виртуална функция се нарича **абстрактен**.

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

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

АБСТРАКТЕН КЛАС

- Предназначение

Абстрактните класове са предназначени да са базови на други класове.

Чрез тях се обединяват в обща структура различни йерархии.

АБСТРАКТЕН КЛАС

- Хетерогенна структура - съставна структура от данни, компонентите на която са от различни типове се нарича хетерогенна.
- Полиморфизмът, с помощта на абстрактните класове, позволява създаването на хетерогенни (полиморфни) структури от данни.

Пример:

Стек, елементите на който са от различен тип:

- точки в равнината и в пространството;
- домашни животни и горски животни;

КРАЙ