

Тема 10

Наследяване. Видове наследяване. Параметри на функции (указатели и референции). Конструктори и деструктори при наследяване.

Наследяване (is-a relationship)

При създаване на нов клас, който има общи компоненти и поведение с вече дефиниран клас, вместо да дефинира повторно тези компоненти и поведение, можем да го обявим за наследник на вече дефинирания клас. При наследяване се наследяват данните и методите на основните класове и се получава обект до избор от наследените компоненти на основните класове

Наследяването се извършва по следния начин:
class B: (тип на наследяване) A

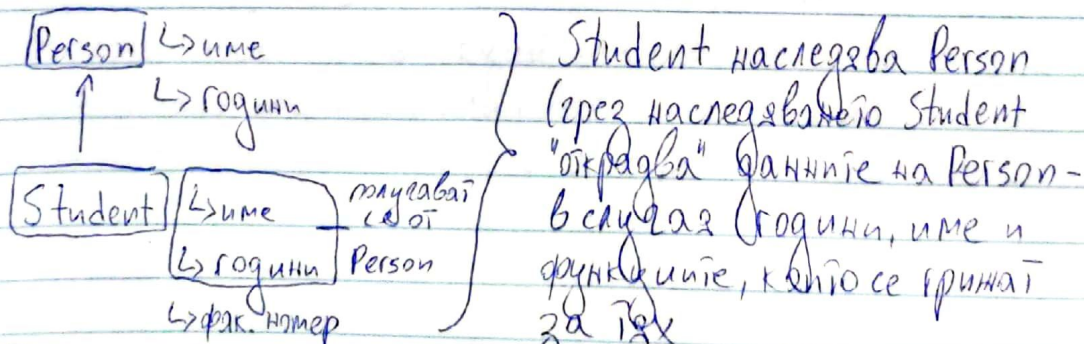
В нашия пример класът B е наследник на клас A. Обектите от класовете изглеждат така:

A

A B

След като B е наследник на клас A, то при създаване на обект от тип B се създава обект от тип A, който е част от обекта от тип B.

Графичен пример:



За данните, които са приети от Person - за тях се принасят Person
За останалите данни, които са в Student се принасят Student

Person

Person p;

name: get() set()
age: 3



Student

Student s;

name: get() set()
age: 3
fn: 1234

Иллюстрация
ген

Типове на наследяване:

- private
- public
- protected

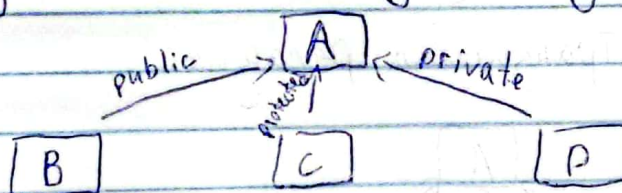
Типът на наследяване може да се пропусне в декларацията на класа/структурата, като по default наследяването на клас е private, а наследяването на структура публич.

Забелешка: Наследяването на приятелски класове не е транзитивна релация, т.е. ако клас A е приятелски клас за клас B и клас B е приятелски клас за клас C, то нито клас A е приятелски клас за клас C, нито клас C е приятелски клас за клас A.

Нека имаме следния клас: и следните видове наследяване:

class A

```
{
    public:
        int x;
    protected:
        int y;
    private:
        int z;
}
```



Тогава наследеният член-данни ще са:

class B: public A

```
{
    x - public
    y - protected
    z - не е достъпен
}
```

class C: protected A

```
{
    x - protected
    y - protected
    z - не е достъпен
}
```

class D: (private) A

```
{
    x - private
    y - private
    z - не е достъпен
}
```

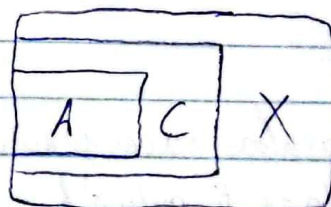
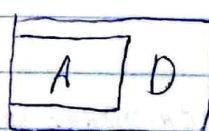
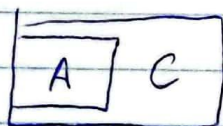
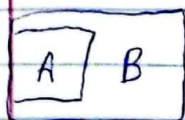
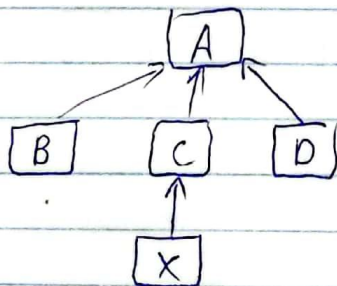

Нека разгледаме следния пример:

class Dog	→	class Person
↳ име		↳ име
↳ години		↳ години
↳ порода		

Тук може да се изкушим да направим наследяване, тъй като Dog и Person има общи атрибут-данни, по подобие на Student и Person. Но в никакъв случай не бива да го правим, защото формално между класове не може да се случи (но като дизайн е грешно). Наследяването между класове се прави, когато между тях има някакво взаимоотношение.

Dog is a Person X (кучето не е човек)
Student is a Person ✓ (ученикът е човек)

Наследяването е транзитивна релация:

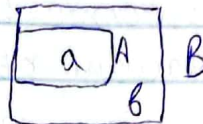


Ако X е наследник на C и C е наследник на A, то X е също наследник на A.

Подаване като параметри във функции
Класове-наследници могат да бъдат подавани като параметри на
функции, които приемат обекти от базовия клас. Може да се използва
функционалността от базовия клас.

Да разгледаме следния пример:

```
class A
{
public:
    int a;
};
class B: public A
```



```
{
    public:
        int b;
};
void f (A& obj)
{
    obj.a++;
}
```

```
void g (A* ptr)
{
    (*ptr).a++;
}
```

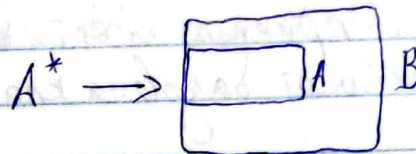
```
void t (B& obj)
{
    obj.b++;
}
```

```
int main()
{
```

```
    A obj1; B obj2;
    f(obj1); // Валидно
    f(obj2); // Валидно
    g(obj1); // Валидно
    g(obj2); // Валидно
    t(obj2); // Валидно
    t(obj1); // Неважно - защото обект от тип A няма елементите на B
```


Нека разгледаме следната ситуация:

```
A obj1;  
B obj2;  
A* ptr1 = &obj1; // Валидно  
A* ptr2 = &obj2; // Валидно  
B* ptr3 = &obj1; // Не валидно  
B* ptr4 = &obj2; // Валидно
```



Можем да насочваме указателите към обекти от ~~типа~~ класове наследници. ~~A* ptr~~ можем да насочим към обект от същия тип (тривиално), както и към обект от тип B, защото в началото на B имаме обект от тип A, B е негов наследник. B* ptr можем да насочим към обект от същия тип, но не и към обект от тип A, защото A не е наследник на B - не съдържа обект от тип B в себе си.

Конструктори и деструктори при наследяване
Във всеки ~~наследник~~ ^{конструктор} на клас-наследник трябва да се оказва кой конструктор да се извика за базовия клас. Ако не е указано, ще се извика default-ния конструктор. Ако базовият клас няма конструктор по подразбиране, то не можем да създадем обект на клас-наследник без да укажем конструктор, който да се извика за базовия клас.

Пример:

```
class B: public A  
{  
    ....  
public:  
    B(...): A(...) // Конструктор на A  
    {  
        // Гржим се само за данните на B  
    }  
};
```


Деструкторът на наследения клас извиква деструктора на базовия клас
~ B()

```
{  
    // Гржим се само за B  
} → извиква деструктора на A
```

Копиране при наследяване

При разписване на конструктор за копиране и оператор: на клас-наследник, трябва експлицитно да извикате копи констр. и оп: и за базовия клас.

```
B(const B& other) : A(other) // к.к. на A
```

```
{  
    copyFrom(other); // гржим се само за данните на B  
}
```

```
B& operator=(const B& other)
```

```
{  
    if (this != &other)  
    {  
        A::operator=(other); // оператор на A  
        free();  
        copyFrom(other);  
    }  
    // гржим се само за данните на B
```

```
    return *this;  
}
```

Разписване копирането и приенето само ако наследникът ползва неинициализирана динамична памет. Ако не използва, то генерираните от компилатора работят коректно.