ОБЕКТНО-ОРИЕНТИРАНО ПРОГРАМИРАНЕ

Магдалина Тодорова

спец. Информационни системи,
I курс

ФМИ, СУ „Св. Климент Охридски“
2018/2019

Тема № 8

НАСЛЕДЯВАНЕ

1. Основни бележки

Наследяването е начин за създаване на нови класове чрез използване на компоненти и поведение на съществуващи класове.

При създаване на нов клас, който има общи компоненти и поведение с вече дефиниран клас, вместо да дефинира повторно тези компоненти и поведение, програмистът може да определи новия клас като клас наследник на вече дефинирания.

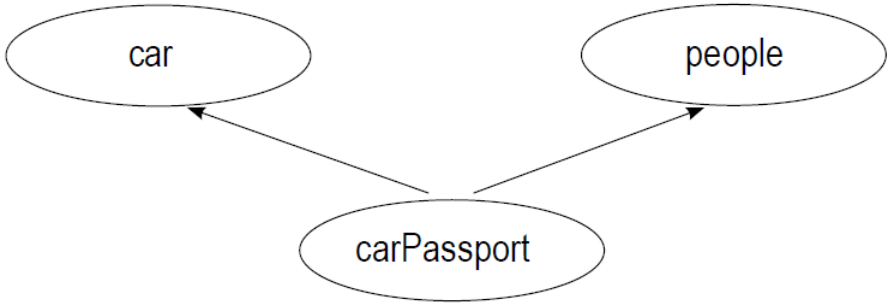
Класът, който е дефиниран първоначално, се нарича ***базов*** или ***основен клас***, а новосъздаденият клас – ***производен***.

1. *Основни бележки*

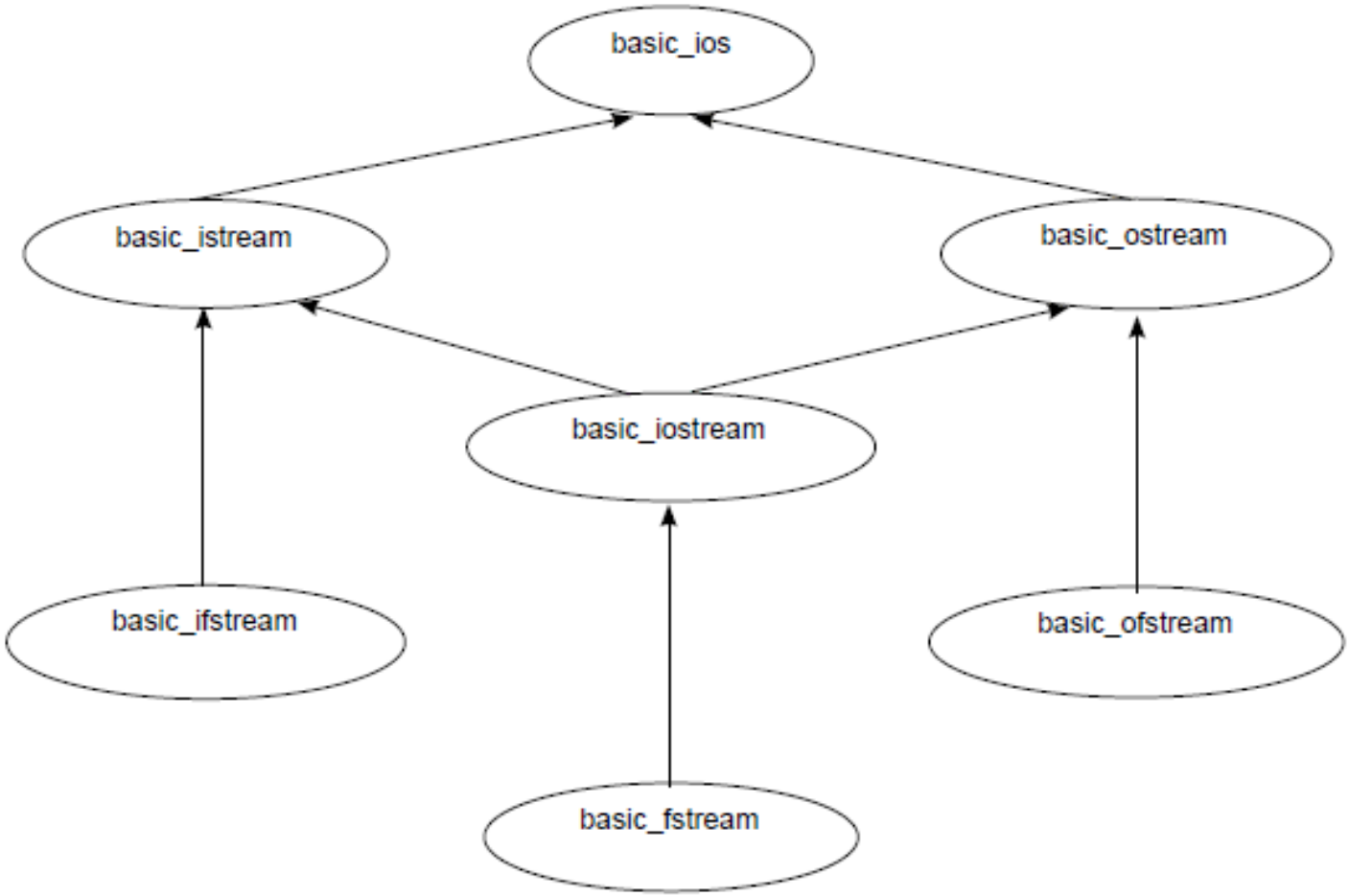
1. *Основни бележки*

Единично наследяване

Множествено наследване



1. *Основни бележки*



Фигура 5.1. Библиотека iostream – шаблони на класове за създаване на потоци

2. Дефиниране 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> **}**

<тяло> ::= **{** <декларация_на_компонента>;

{ <декларация_на_компонента>; **}**

} [<списък_от_обекти> **];**

2. *Дефиниране на производен клас*

<декларация_на_компонента> ::=

<декларация_на_конструктор> |

<декларация_на_мутатор> |

<декларация_на_функция_за_достъп> |

<декларация_на_член_данна>

2. *Дефиниране на производен клас*

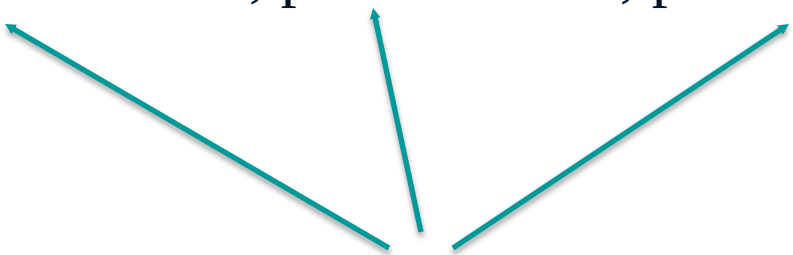
<име_на-производен_клас> ::= <идентификатор>

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

<име_на_базов_клас> ::= <идентификатор>

Примери.

```
class der : public base1, private base2, protected base3
{ ...
};
```



2. *Дефиниране на производен клас*

Декларацията

```
class der : base1, base2, base3
{ ...
};
```

е еквивалентна на

```
class der : private base1, private base2, private base3
{ ...
};
```

2. *Дефиниране на производен клас*

```
class der : public base1, base2, base3
{ ...
};
```

е еквивалентна на

```
class der : public base1, private base2, private base3
{ ...
};
```

```
class der : protected base1, base2, public base3
{ ...
};
```

е еквивалентна на

```
class der : protected base1, private base2, public base3
{ ...
};
```

2. *Дефиниране на производен клас*

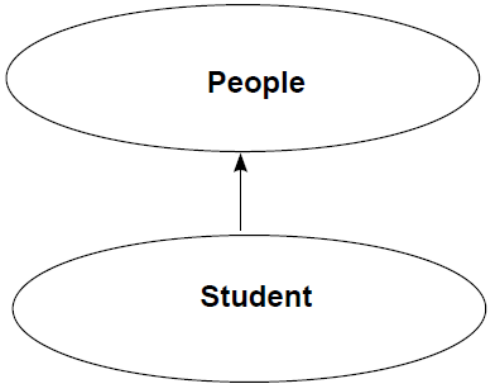
Дефинирането на методите на производен клас се осъществява по същия начин като дефинирането на методи на обикновени класове.

2. *Дефиниране на производен клас - пример*

Задача.

Да се напише програма, която дефинира клас *People*, определящ човек по *име* и *единен граждански номер* (ЕГН), а също производен клас *Student* на класа *People*, определящ студент като ***човек***, който има *факултетен номер* и *среден успех*.

Да се дефинира обект от клас *Student* и се изведе дефинираният обект.



2. *Дефиниране на производен клас - пример*

```
#include <iostream>
#include <cstring>
#include <cassert>
using namespace std;
// дефиниция на базовия клас People
class People
{ public:
      void ReadPeople(char*, char*);
      void PrintPeople() const;
   private:
      char* name;                  // име
      char* ucn;                   // ЕГН (uniform civil number)
};
```

2. *Дефиниране на производен клас - пример*

```
void People::ReadPeople(char* na, char* uc)
{ name = new char[strlen(na)+1];
  assert(name != NULL);
  strcpy(name, na);        // strcpy_s
  ucn = new char[strlen(uc)+1];
  assert(ucn != NULL);
  strcpy(ucn, uc);         // strcpy_s
}

void People::PrintPeople() const
{ cout << "Name: " << name << endl;
  cout << "UCN: " << ucn << endl;
}
```

2. *Дефиниране на производен клас - пример*

```
// дефиниция на производния клас Student

class Student : public People
{ public:
      void ReadStudent(char*, char*, long, double);
      void PrintStudent() const;
   private:
      long fac_numb;              // факултетен номер
      double gpa;                 // среден успех (grade point average)
};
```

2. *Дефиниране на производен клас - пример*

```
// може да си мислим Student като клас от вида:
class Student
{ public:
      void ReadPeople(char*, char*);
      void PrintPeople() const;
      void ReadStudent(char*, char*, long, double);
      void PrintStudent() const;
   private:
      char* name;              // име
      char* ucn;               // ЕГН
      long fac_numb;           // факултетен номер
      double gpa;              // среден успех
};
```

2. *Дефиниране на производен клас - пример*

Освен това собствените методи на класа *Student* нямат пряк достъп до ***private*** компонентите на класа *People*.

```
void Student::ReadStudent(char* na, char* uc,
                                    long facn, double g)
 // ReadStudent няма пряк достъп до name и ucn
{ ReadPeople(na, uc); // непряк достъп до name и ucn
   fac_numb = facn;
   gpa = g;
}
```

2. *Дефиниране на производен клас - пример*

```
void Student::PrintStudent() const
// PrintStudent няма пряк достъп до name и ucn
{ PrintPeople();    // непряк достъп до name и ucn
  cout << "Fac. number: " << fac_numb << endl;
  cout << "GPA of student: " << gpa << endl;
}
```

2. *Дефиниране на производен клас - пример*

```
int main()
{ Student stud;
  stud.ReadStudent("Ivan Ivanov", "9206123422",
                                  48444, 5.0);
  stud.PrintStudent();
  return 0;
}
```

2. *Дефиниране 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> **}**

2. *Дефиниране на производен клас*

<тяло> ::= { <декларация_на_компонента>;
{ <декларация_на_компонента>; **}**
} **[** <списък_от_обекти> **]**;

<декларация_на_компонента> ::=
<декларация_на_конструктор> |
<декларация_на_мутатор> |
<декларация_на_функция_за_достъп> |
<декларация_на_член_данна>

2. *Дефиниране на производен клас*

<име_на-производен_клас> ::= <идентификатор>

<атрибут_за_област> ::= public | protected | private

<име_на_базов_клас> ::= <идентификатор>

Множеството от компонентите на производен клас се състои от компонентите на неговите базови класове и компонентите, декларирани в самия производен клас. Оттук произлиза и терминът наследяване.
Механизмът, чрез който производният клас получава компонентите на базовия, се нарича **наследяване**.

2. *Дефиниране на производен клас*

Когато производният клас има няколко базови класа, той наследява компонентите на всеки от тях. Наследяването в този случай е ***множествено***.

2. *Дефиниране на производен клас*

Процесът на наследяване се изразява в следното:

– наследяват се член-данните и методите на основните класове;

– получава се достъп до някои от наследените компоненти на основните класове;

– производният клас „познава“ реализациите само на основните класове, от които произлиза;

– производният клас може да е основен за други класове.

2. *Дефиниране на производен клас*

Производният клас може да дефинира допълнително:

– свои член-данни;

– свои член-функции (методи), аналогични на тези на основните класове, а също и нови.

Дефинираните в производния клас член-данни и член-функции се наричат ***собствени***.

2. *Дефиниране на производен клас*

Основните класове могат да са ***директни*** и ***индиректни*** основни класове на производен клас.

Директните основни класове се изброяват в заглавието на производния клас, предшествани от двоеточие **:**.

Например класът *People* от предходната задача е директен основен клас на класа *Student*.

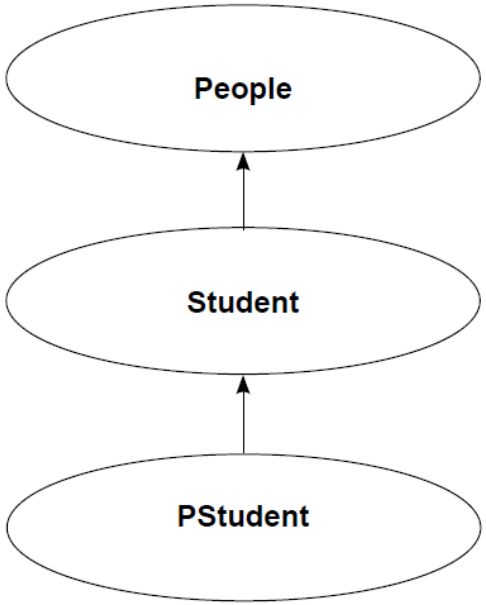
Индиректните основни класове не се изброяват в заглавието на производните класове, но се наследяват чрез директните основни клсове.

2. *Дефиниране на производен клас - пример*

*Пример за **индиректен** основен клас*

Задача.

Да се дефинира клас *PStudent*, производен на класа *Student*, реализиращ студент в платена форма на обучение.



2. *Дефиниране на производен клас - пример*

```
class PStudent : public Student
{ public:
      void ReadPStudent(char*, char*, long, double, double);
      void PrintPStudent() const;
   private:
      double fee;          // такса за обучение
};
```

2. *Дефиниране на производен клас - пример*

Може да мислим класа Pstudent като клас от вида

```
class PStudent
{ public:
    void ReadPeople(char*, char*);
     void PrintPeople() const;
     void ReadStudent(char*, char*, long, double);
     void PrintStudent() const;
     void ReadPStudent(char*, char*, long, double, double);
     void PrintPStudent() const;
   private:
      char* name;            // име
      char* ucn;             // ЕГН
      long fac_numb;         // факултетен номер
      double gpa;            // среден успех
      double fee;            // такса за обучение
};
```

2. *Дефиниране на производен клас - пример*

```
void PStudent::ReadPStudent(char* na, char* uc, long facn,
                                        double g, double f)
{  // задаване на стойности за наследените член-данни
   ReadStudent(na, uc, facn, g);
   // задаване на стойност на собствената член-данна
   fee = f;
}
```

2. *Дефиниране на производен клас - пример*

```
void PStudent::PrintPStudent() const
{ // извеждане на стойностите за наследените член-данни
  PrintStudent();
  // извеждане на стойността на собствената член-данна
  cout << "Fee: " << fee << endl;
}
```

2. *Дефиниране на производен клас - пример*

Ако е дефиниран обектът pstud

PStudent pstud;

pstud има достъп до

```
pstud.ReadPStudent("Ivan Ivanov", "9206123422", 48444, 5.0,
                                                    3000);
pstud.PrintPStudent();
pstud.ReadStudent("Ivan Ivanov", "9206123422", 48444, 5.0);
pstud.PrintStudent();
pstud.ReadPeople("Ivan Ivanov", "9206123422");
pstud.PrintPeople();
```

3. Единично наследяване

Атрибут за област	Компонента на основен клас, определена в секция	Наследява се като
public	public protected private	public protected private
private	public protected private	private private private
protected	public protected private	protected protected private

3. *Единично наследяване*

От таблицата се вижда, че:

• Ако базовият клас е деклариран с атрибут за област ***public*** в производния клас, *public*, *protected* и *private* компонентите на базовия клас се наследяват съответно като *public*, *protected* и *private* компоненти на производния клас.

Пример. Ако

```
class base
{ public: int b3();
  protected: int b2;
  private: int b1;
};
```

```
class der1 : public base
{ public: int d3();
  protected: int d2;
  private: int d1;
};
```

3. *Единично наследяване*

можем да мислим, че *der1* е клас от вида

```
class der1
{ public:
      int b3();
      int d3();
   protected:
      int b2;
      int d2;
   private:
      int b1;
      int d1;
};
```

3. *Единично наследяване*

• Ако базовият клас е деклариран с атрибут за област ***private*** в производния клас, всички негови компоненти се наследяват като *private*.

Пример. Ако

```
class base
{ public: int b3();
  protected: int b2;
  private: int b1;
};
```

```
class der2 : private base
{ public: int d3();
  protected: int d2;
  private: int d1;
};
```

3. *Единично наследяване*

можем да мислим, че *der2* е клас от вида

```
class der2
{ public:
      int d3();
  protected:
      int d2;
  private:
      int b3();
      int b2;
      int b1;
      int d1;
};
```

3. *Единично наследяване*

• Ако базовият клас е деклариран с атрибут за област ***protected*** в производния клас, *private* компонентите му се наследяват като *private*, а *public* и *protected* – като *protected*.

Пример. Ако

```
class base
{ public: int b3();
  protected: int b2;
  private: int b1;
};
```

```
class der3 : protected base
{ public: int d3();
  protected: int d2;
  private: int d1;
};
```

3. *Единично наследяване*

можем да мислим, че *der3* е клас от вида

```
class der3
{ public:
      int d3();
  protected:
      int b3();
      int b2;
      int d2;
   private:
      int b1;
      int d1;
};
```

3. *Единично наследяване*

Наследените компоненти се различават от собствените компоненти на производния клас по правата за достъп.

Както при обикновените класове (класовете без наследяване), ***собствените компоненти на производния клас имат пряк достъп помежду си.***

3. *Единично наследяване*

Собствените компоненти на производния клас имат пряк достъп до компонентите, декларирани като **public** и **protected** в основния му клас и нямат пряк достъп до декларираните като **private** компоненти на основния му клас.

Достъпът до **private** компонентите на основния клас може да се извърши чрез неговия интерфейс.

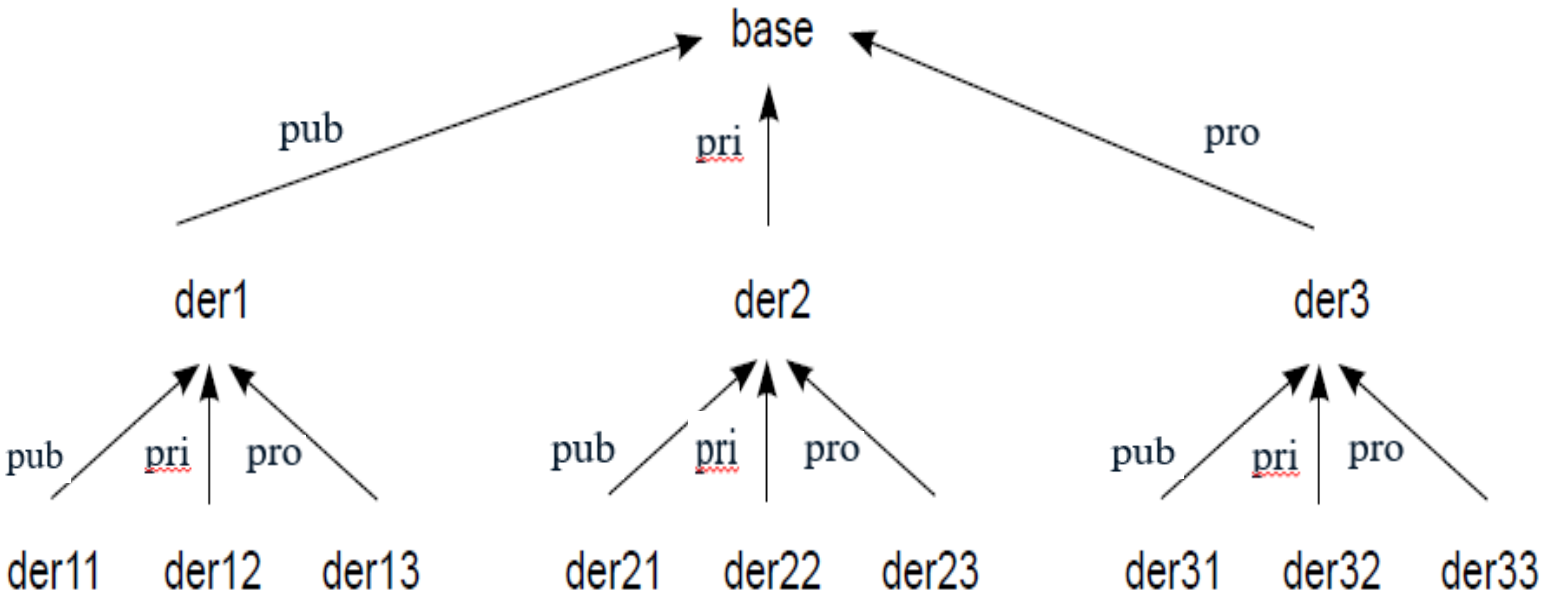
3. *Единично наследяване*

Таблицата показва локалния достъп (нарича се още ***вътрешен*** или ***пряк достъп*** **ПД**) на член-функции на производен клас до компонентите на базовия му клас и ***външния достъп*** (**ВД**) на обект на производен клас до компонентите на базовия му клас.

компонента на базов клас	производен клас - атрибут **public** на базовия му клас		производен клас - атрибут **private** на базовия му клас		производен клас - атрибут **protected** на базовия му клас	
	ПД	ВД	ПД	ВД	ПД	ВД
public	да	да	да	не	да	не
protected	да	не	да	не	да	не
private	не	не	не	не	не	не

3. *Единично наследяване*

Пример.



3. *Единично наследяване*

Пример.

```
class base
{ public: int a3();
  protected: int a2;
  private: int a1;
} b;
```

```
class der1 : public base
{ public: int a6();
  protected: int a5;
  private: int a4;
} d1;
```

```
class der2 : private base
{ public: int a9();
  protected: int a8;
  private: int a7;
} d2;
```

```
class der3:protected base
{ public: int a12();
  protected: int a11;
  private: int a10;
} d3;
```

```
class der11 : public der1
{ public: int a15();
  protected: int a14;
  private: int a13;
} d4;
```

```
class der12 : private der1
{ public: int a18();
  protected: int a17;
  private: int a16;
} d5;
```

```
class der13:protected der1
{ public: int a21();
  protected: int a20;
  private: int a19;
} d6;
```

```
class der21 : public der2
{ public: int a24();
  protected: int a23;
  private: int a22;
} d7;
```

```
class der22 : private der2
{ public: int a27();
  protected: int a26;
  private: int a25;
} d8;
```

```
class der23:protected der2
{ public: int a30();
  protected: int a29;
  private: int a28;
} d9;
```

```
class der31 : public der3
{ public: int a33();
  protected: int a32;
  private: int a31;
} d10;
```

```
class der32 : private der3
{ public: int a36();
  protected: int a35;
  private: int a34;
} d11;
```

```
class der33:protected der3
{ public: int a39();
  protected: int a38;
  private: int a37;
} d12;
```

3. *Единично наследяване*

Правила за достъп

• ***Достъп до компоненти на основен клас чрез методи на производен клас*** (*без значение на атрибута за област*)

*- Методите на производен клас **нямат** пряк достъп до **private** компонентите (член-данни и член-функции) на основния му клас.*

*- Методите на производен клас **имат** пряк достъп до **public** и **protected** компонентите (член-данни и член-функции) на основния му клас.*

3. *Единично наследяване*

Правила за достъп

• ***Достъп до компоненти чрез обекти на основния и производния класове (ВД)***

- *Обект на основен клас има* ***достъп*** *до всички свои компоненти, обявени като* **public** *и* ***няма достъп*** *до компонентите, обявени като* **private** *и* **protected**.

- *Обект на производен клас има достъп до* **public** *компонентите на класа, от който е и до компонентите на основния клас, наследени в производния клас като* **public**.

Правила за достъп

- ***Достъп на методи на основен клас до компоненти (член-данни и методи) на производен клас***

Методите на основния клас нямат достъп до компонентите на производен на него клас. Причината е, че когато основният клас се дефинира, не е ясно какви производни класове ще произхождат от него.

Допустими са редица присвоявания между обекти, указатели към обекти и псевдоними на обекти на основния и производния клас. За реализирането им се извършват преобразувания.

3. *Единично наследяване*

• ***Достъп на функции-приятели на производен клас до компоненти на основния му клас***

- Функциите-приятели на производен клас имат същите права на достъп като на член-функциите на производния клас, т.е. ***имат пряк достъп до всички компоненти на производния клас и до public и protected компонентите на основния му клас*.**

- ***Декларацията за приятелство не се наследява*.** Функция-приятел на основен клас не е приятел на производния му клас, освен ако не е декларирана като такава.

3. *Единично наследяване*

Пример.

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

class base
{ public:
      void readbase(int x, int y)
      { a1 = x;
        a2 = y;
      }
      void a3() const
      { cout << "a1: " << a1 << endl
             << "a2: " << a2 << endl;
      }
```

3. *Единично наследяване*

```
    protected: int a2;
    private: int a1;
};
class der : public base
{ public:
      friend void f(der& d, int x, int y, int z)
      { cout << "friend function f(): \n";
        d.d1 = x;  // достъп до компонента на класа der
        d.d2 = y;  // достъп до компонента на класа der
        d.a2 = z;  // достъп до protected компонента на base
        // cout << "d.a3(): " << endl;
        d.a3();    // достъп до public компонента на base
        cout << "d.d3(): " << endl;
        d.d3();    // достъп до компонента на класа der
      }
```

3. *Единично наследяване*

```
    void readder(int x, int y, int z, int t)
    { readbase(x, y);
      d1 = z;
      d2 = t;
    }
    void d3() const
    { cout  << "d1: " << d1 << endl
            << "d2: " << d2 << endl
            << "a2: " << a2 << endl;
      cout << "a3():" << endl;
      a3();
    }
  protected: int d2;
  private: int d1;
};
```

3. *Единично наследяване*

```
int main()
{ der x;
   x.readder(10, 20, 30, 40);
   x.d3();
   f(x, 100, 200, 300);
   return 0;
}
```

се получава

```
d1: 30
d2: 40
a2: 20
a3():
a1: 10
a2: 20
friend function f():
d.a3():
a1: 10
a2: 300
d.d3():
d1: 100
d2: 200
a2: 300
a3():
a1: 10
a2: 300
```

4. Предефиниране на компоненти

Един проблем при наследяването е, че производният клас може да наследи член-функция, която не трябва да има. Решението на този проблем се осъществява като в производния клас се предефинира (дефинира се повторно) член-функцията с подходяща реализация.

4. *Предефиниране на компоненти*

Базовият и производният му клас могат да притежават собствени компоненти с еднакви имена. В този случай производният клас ще притежава компоненти с еднакви имена. Обръщението към такава компонента чрез обект от производния клас извиква собствената на производния клас компонента, т.е. ***името на собствената компонента е с по-висок приоритет от това на наследената***.

За да се изпълни наследена компонента се указва пълното й име, т.е.

<име_на_основен_клас>::<компонента>

4. Предефиниране на компоненти

Пример.

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

class base
{ public:
      void init(int x)
      { bx = x;
      }
      void display() const
      { cout << " class base: bx= " << bx << endl;
      }
   protected:
      int bx;
};
```

4. *Предефиниране на компоненти*

```
class der : public base
{ public:
      void init(int x)
      { bx = x;
        base::bx = x + 5;
      }
      void display() const
      { cout << " class der: bx = " << bx;
        cout << " base::bx = " << base::bx << endl;
      }
   protected:
      int bx;
};
```

4. *Предефиниране на компоненти*

```
int main()
{ base b;
  der d;
  b.init(5);              // обръщение към base::init
  d.init(10);             // обръщение към der::init
  b.display();            // обръщение към base::display
  d.display();            // обръщение към der::display
  d.base::init(20);       // обръщение към base::init
  d.base::display();      // обръщение към base::display
  d.display();            // обръщение към der::display
  b.display();            // обръщение към base::display
  return 0;
}
```

4. *Предефиниране на компоненти*

В резултат от изпълнението й се получава:

class base: bx = 5
class der: bx = 10 base::bx = 15
class base: bx = 20
class der: bx = 10 base::bx = 20
class base: bx = 5