Обектно ориентирано програмиране

НАСЛЕДЯВАНЕ. ПРОИЗВОДНИ КЛАСОВЕ

Наследяване.
Производни класове

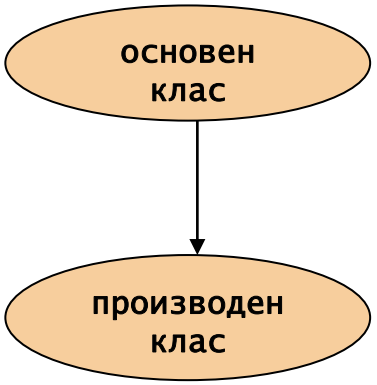
Производните класове и наследяването са една от най-важните характеристики на обектно-ориентираното програмиране (ООП).

Чрез механизма на наследяване от съществуващ клас се създава нов клас. Класът от който се създава се нарича **базов (основен) клас**, а този, който е създаден - **производен**.

Наследяване. Производни класове ...

Понятията основен и производен клас са относителни, тъй като производен клас може да е основен за други класове, а основен – да е производен от други основни класове.

Производният клас може да наследи компонентите на един или няколко базови класа. В първия случай наследяването се нарича **единично (просто)**, а във втория – **множествено**.



Наследяване. Производни класове ...

Дефинирането на производни класове е еквивалентно на конструирането на йерархии от класове.

Защо се налага дефинирането на производни класове?

В кои случаи и как се прави това?

Какви са предимствата от дефинирането на производни класове?

На тези въпроси ще дадем отговор в следващите разглеждания.

Наследяване. Производни класове ...

Ако множество от класове имат общи данни и методи, тези общи части могат да се обособят като основни класове, а всяка от останалите части да се дефинира като производен клас на съответния основен клас. Така се прави икономия на памет, тъй като се избягва многократното описание на едни и същи програмни фрагменти.

При конструирането на производни класове е достатъчно да се разполага само с обектните модули на основните класове, а не с техния програмен код. Това позволява да бъдат създавани библиотеки от класове, които да бъдат използвани при създаването на производни класове.

Тези предимства, а също възможността за реализиране на полиморфизъм, мотивират въвеждането на производни класове.

Дефиниране на производни класове

Подобно на обикновените, производните класове се дефинират като се *декларира класът* и се *дефинират неговите методи*.

Деклариране на производен клас

<декларация_на_производен_клас> ::=

class <име_на-производен_клас> :

[<атрибут_за_област>] <име_на_базов_клас>

{, [<атрибут_за_област>] <име_на_базов_клас>}

{<декларация_на_компоненти>

};

<име_на-производен_клас> ::= <идентификатор>

<атрибут_за_област> ::= public | private | protected

<име_на_базов_клас> ::= <идентификатор>

Дефиниране на производни класове ...

Пред всяко име на базов клас *може* да се постави запазената дума public, private или protected. Нарича се **атрибут за област**, тъй като определя областта на наследените членове. Употребата на атрибутите за област е различна от тази за обявяване на секции в тялото на класа. Ако атрибут за област е пропуснат, подразбира се private. Атрибутът protected е включен в новите версии на езика и не се използва много често.

Примери:

1.

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

Дефиниране на производни класове ...

Тъй като атрибутът за област е пропуснат и за трите базови класа, подразбира се private, т.е. декларацията е еквивалентна на

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

Дефиниране на производни класове …

2. Декларацията:

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

е еквивалентна на

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

Дефиниране на производни класове …

3. Декларацията:

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

е еквивалентна на

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

Декларациите на компонентите на производен клас, а също дефинициите на неговите методи не се различават от съответните при обикновените класове.

Дефиниране на производни класове ...

Множеството от компонентите на един 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.

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

Дефиниране на производни класове …

Процесът на наследяване се изразява в следното:

- наследяват се данните и методите на основния клас;
- получава се достъп до някои от наследените членове на основния клас;
- производният клас “познава” реализацията само на основния клас, от който произлиза;
- производният клас може да е основен за други класове.

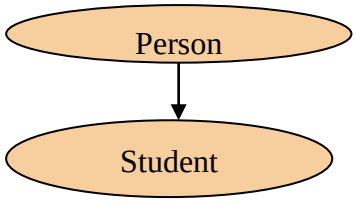
Производният клас може да дефинира допълнително:

- свои член-данни;
- методи, аналогични на тези на основния клас, а също и нови.

Дефинираните в производния клас данни и методи се наричат **собствени**.

Дефиниране на производни класове ...

Задача. Да се напише програма, която дефинира клас Person, определящ човек по име и единен граждански номер (ЕГН), а също производен клас Student на класа Person, който определя понятието студент като човек, който има факултетен номер и среден успех. Да се дефинира обект от клас Student и се изведе дефинираният обект.



Дефиниране на производни класове …

```
/* Person.h */
#pragma once
// дефиниция на базовия клас Person
class Person {
public:
   void readPerson(char *, char *);
   void printPerson() const;
private:
   char * name;
   char * egn;
};
```

Дефиниране на производни класове …

```
#include "Person.h"
#include <string>
#include <iostream>
using namespace std;
void Person::readPerson(char *str, char *num)
{
   name = new char[strlen(str) + 1];
   strcpy(name, str);
   egn = new char[11];
   strcpy(egn, num);
}
void Person::printPerson() const
{
   cout << "Ime: " << name << endl;
   cout << "EGN: " << egn << endl;
}
```

Дефиниране на производни класове …

Чрез класа Person е представено понятието човек, характеризиращо човек с име и ЕГН, реализирани чрез член-данните **name** и **egn** от тип **char***. Капсулирани са чрез декларирането им като **private**.

Освен член-данни класът съдържа и методите **readPerson** и **printPerson**, образуващи интерфейса на класа (обявени са като **public**).

Чрез **readPerson** се инициализират обектите на класа Person, а чрез **printPerson** се извеждат върху екрана стойностите на член-данните name и egn. Ще напомним, че заделената от методите динамична памет не се освобождава автоматично при унищожаване на обектите. Освобождаването на тази памет трябва да стане явно, чрез оператора delete. В тази част умишлено методът ReadPerson не е реализиран като конструктор, не е дефиниран също и деструктор.

Дефиниране на производни класове …

```
/* Student.h */
#pragma once
#include "Person.h"
class Student : Person
{
public:
   void readStudent(char *, char *, long, double);
   void printStudent() const;
private:
   long facnom;
   double usp;
};
```

Дефиниране на производни класове …

```
/* Student.cpp */
#include "Student.h"
#include <string>
#include <iostream>
using namespace std;
void Student::readStudent(char *str, char * num, long facn, double u)
{
   readPerson(str, num);
   facnom = facn;
   usp = u;
}
void Student::printStudent() const
{
   printPerson();
   cout << "fac. nomer: " << facnom << endl;
   cout << "uspeh: " << usp << endl;
}
```

Дефиниране на производни класове …

```
/* Main.cpp */
#include "Student.h"

int main()
{
   Student stud;
   stud.readStudent("Ivan Ivanov", "8206123422",
                    42444, 6.0);
   stud.printStudent();
   return 0;
}
```

Дефиниране на производни класове …

Класът Student, дефиниран в програмата, представя понятието студент, като реализира следното определение: Студент, това е човек, който има факултетен номер и се характеризира със среден успех. Това дава основание Student да бъде определен като производен клас на класа Person.

В резултат, класът Student има осем компоненти. Четири от тях (name, egn, readPerson и printPerson) са наследени от базовия клас Person и четири (facnom, usp, readStudent и printStudent) са декларирани в него.

Тъй като не е указан атрибут за област на базовия клас Person, подразбира се private. В този случай производният клас Student наследява всички член-данни и член-функции на основния клас като private. Освен това той получава възможността да използва всички компоненти на основния клас, които не са private (в случая readPerson и printPerson). Така член-функциите readStudent и printStudent нямат пряк достъп до наследените член-данни name и egn. Затова достъпът е реализиран чрез методите readPerson и printPerson.

Дефиниране на производни класове ...

Производният клас е дефиниран след като вече е дефиниран базовият клас, от който той произлиза. Чрез него се разширява декларацията на съществуващ клас. Разширяемостта на класовете е една от важните характеристики на ООП.

Чрез следващата задача ще покажем възможността производен клас да е основен за друг клас, т.е. да бъде създадена верига от наследени класове.

Задача. Да се дефинира клас PStudent, производен на класа Student, реализиращ понятието студент от платена форма на обучение.

Дефиниране на производни класове …

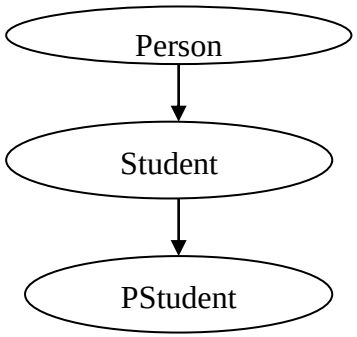
```
#pragma once

#include "Student.h"

class PStudent :

public Student

{

public:

   void readPStudent(char *, char *, long, double, double);

   void printPStudent() const;

private:

   double tax; // такса за обучението на студента

};
```

Дефиниране на производни класове …

```
#include "PStudent.h"
#include <iostream>
using namespace std;
void PStudent::readPStudent(char *str, char *num, long facn, double u, double t)
{
   readStudent(str, num, facn, u);
   tax = t;
}
void PStudent::printPStudent() const
{
   printStudent();
   cout << "Tax: " << tax << endl;
}
```

Дефиниране на производни класове ...

Класът PStudent е производен на класа Student с атрибут за област public. В този случай PStudent наследява всички компоненти на класа Student (собствени и наследени от Person) като запазва вида им, т.е. собствените методи readStudent и printStudent продължават да са public, а собствените член-данни facnom и usp и всички наследени от Person продължават да са private в класа PStudent. Това е така, тъй като атрибутът за област на класа Student е private, заради което всички компоненти на Person са наследени от Student като private и отново като private се наследяват и от класа PStudent.



Наследяване и достъп до наследените компоненти

Ще напомним, че в рамките на един клас (без наследяване), protected частта има аналогична роля като тази на private частта. До компоненти от тип protected имат пряк достъп само член-функции и приятелски функции на класа.

Атрибутът за област на базовия клас в декларацията на производния клас (public, private или protected) управлява механизма на наследяване и определя какъв да бъде режимът на достъп до наследените членове.

Наследяване и достъп до наследените компоненти …

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

Наследявания на компоненти на основен клас в производен

Наследяване и достъп до наследените компоненти ...

Ако базовият клас е деклариран като public в производния клас, всички private, public и protected компоненти на базовия клас се наследяват съответно като private, public и protected компоненти на производния клас.

Пример: Ако

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

Наследяване и достъп до наследените компоненти …

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

можем да си мислим, че der1 е клас от вида:

Наследяване и достъп до наследените компоненти …

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

Наследяване и достъп до наследените компоненти ...

Ако базовият клас е деклариран като private в производния клас, всички негови компоненти се наследяват като private.

Пример: Ако

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

Наследяване и достъп до наследените компоненти …

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

можем да си мислим, че der2 е клас от вида:

Наследяване и достъп до наследените компоненти …

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

Наследяване и достъп до наследените компоненти ...

Ако базовият клас е деклариран като protected в производния клас, private компонентите му се наследяват като private, а public и protected – като protected.

Пример: Ако

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

Наследяване и достъп до наследените компоненти ...

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

можем да си мислим, че der3 е клас от вида:

Наследяване и достъп до наследените компоненти …

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

Наследяване и достъп до наследените компоненти ...

Наследените компоненти обаче се различават от декларираните в производния клас по правата за достъп. Производният клас има пряк достъп до компонентите, декларирани като public и protected, но няма пряк достъп до декларираните като private в базовия клас.

Достъпът до private компонентите на базовия клас се извършва чрез неговия интерфейс.

Следващата таблица показва прекия достъп на член-функции на производния клас (ПД) и външния достъп на производния клас (ВД) до компонентите на базовия клас.

Наследяване и достъп до наследените компоненти ...

Компонента на базов клас	**Произв. клас с атрибут public**		**Произв. клас с атрибут private**		**Произв. клас с атрибут protected**	
	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

Достъп до компонентите на базовия клас

Наследяване и достъп до наследените компоненти ...

Да се върнем към означенията от последните три примера.

Собствените компоненти на класа der1 са видими навсякъде в класа. Те имат пряк достъп до компонентите b2 и b3() на base, но нямат пряк достъп до private-компонентата b1 на base.

Същото се отнася и за класовете der2 и der3. Освен това:

- обект от клас der1 има пряк достъп public-компонентите b3() – наследена и d3() – собствена за der1;
- обект от клас der2 има пряк достъп единствено до собствената public-компонента d3(), тъй като всички наследени от base компоненти се наследяват като private

Наследяване и достъп до наследените компоненти ...

- обект от клас der3 има също пряк достъп единствено до собствената public-компонента d3(), тъй като public и protected компонентите на base се наследяват като protected в der3.

Ще се спрем на някои от често срещаните случаи за достъп до членове на производен и основен клас, а също на достъпа на външни функции до наследен компонент. Ще изкажем и някои правила за достъп до компоненти на базови и производни класове, които ще подкрепим с още примери.

Наследяване и достъп до наследените компоненти ...

Достъп до членове на основен клас чрез дефиниции на методи на производен клас

В сила са следните правила за достъп:

- *Методите на производен клас (без значение на атрибута за област) нямат директен достъп до членовете от private–секцията на основния му клас*

Наследяване и достъп до наследените компоненти ...

Примери:

а) Класът Student е производен на класа Person. Атрибутът за област не е указан явно, заради което се подразбира private. Методите на Student нямат пряк достъп до private членовете name и egn на Person.

б) Класът PStudent е производен на Student. Атрибутът за област е public. Видът на наследените секции на Student се запазва. Методите на PStudent нямат пряк достъп както до собствените private компоненти facnom и usp на Student, така и до наследените от Person private членовете name и egn.

Ще отбележим също,че тъй като атрибутът за област на класа Person в Student е private, всички компоненти на Person са private в Student и са недостъпни пряко в PStudent.

Наследяване и достъп до наследените компоненти ...

- *В дефинициите на собствени методи на производния клас могат да се използват методите от секциите public и protected на основния му клас*

Примери:

а) Тъй като методите на Student нямат пряк достъп до name и egn, инициализацията на тези компоненти в readStudent става чрез метода readPerson на класа Person, който е обявен в public секцията на Person.

б) Тъй като методите на PStudent нямат пряк достъп до facnom, usp, name и egn, инициализацията им в readPStudent става чрез метода readStudent на класа Student, който е обявен в public секцията на Student.

Наследяване и достъп до наследените компоненти ...

- *В дефинициите на собствени методи на производния клас може директно да се използват член-данните на секцията protected на основния му клас*

Ще илюстрираме това правило като извършим промени в програмата за студенти.

Задача. Да се промени програмата така, че освен класовете Person и Student да включва и наследения от Student клас PStudent. Освен това, методите на производните класове да могат пряко да използват наследените член-данни на основните им класове.

Наследяване и достъп до наследените компоненти …

```
/* Person.h */
#pragma once
// дефиниция на базовия клас Person
class Person
{
public:
   void readPerson(char *, char *);
   void printPerson() const;
protected:// вместо private:
   char * name;
   char * egn;
};
```

Наследяване и достъп до наследените компоненти …

```
#include "Person.h"
#include <string>
#include <iostream>
using namespace std;
void Person::readPerson(char *str, char *num)
{
   name = new char[strlen(str) + 1];
   strcpy(name, str);
   egn = new char[11];
   strcpy(egn, num);
}
void Person::printPerson() const
{
   cout << "Ime: " << name << endl;
   cout << "EGN: " << egn << endl;
}
```

Наследяване и достъп до наследените компоненти …

```
/* Student.h */
#pragma once
#include "Person.h"
class Student : public Person // вместо private:
{
public:
   void readStudent(char *, char *, long, double);
   void printStudent() const;
protected: // вместо private
   long facnom;
   double usp;
};
```

Наследяване и достъп до наследените компоненти …

```
/* Student.cpp */
#include "Student.h"
#include <string>
#include <iostream>
using namespace std;
void Student::readStudent(char *str, char * num, long facn, double u)
{
   name = new char[strlen(str) + 1]; // използваме член-данните
   strcpy(name, str);                // name и egn
   egn = new char[11];
   strcpy(egn, num);
   facnom = facn;
   usp = u;
}
```

Наследяване и достъп до наследените компоненти …

```
void Student::printStudent() const
{
   // използваме член-данните name и egn
   cout << "Ime: " << name << endl;
   cout << "EGN: " << egn << endl;
   cout << "fac. nomer: " << facnom << endl;
   cout << "uspeh: " << usp << endl;
}
```

Наследяване и достъп до наследените компоненти …

```
#pragma once
#include "Student.h"
class PStudent : public Student
{
public:
   void readPStudent(char *, char *, long, double, double);
   void printPStudent() const;
private:
  double tax; // такса за обучението на студента
};
```

Наследяване и достъп до наследените компоненти …

```
#include "PStudent.h"
#include <string>
#include <iostream>
using namespace std;
void PStudent::readPStudent(char *str, char *num,
long facn, double u, double t)
{
   // пряк достъп до
   // name, egn, facnom и usp
   name = new char[strlen(str) + 1];
   strcpy(name, str);
   egn = new char[11];
   strcpy(egn, num);
   facnom = facn;
   usp = u;
   tax = t;
}
```

Наследяване и достъп до наследените компоненти …

```
void PStudent::printPStudent() const
{
    cout << "Ime: " << name << endl;
    cout << "EGN: " << egn << endl;
    cout << "fac. nomer: " << facnom << endl;
    cout << "uspeh: " << usp << endl;
    cout << "Tax: " << tax << endl;
}
```

Наследяване и достъп до наследените компоненти …

```
#include "PStudent.h"
int main()
{
    PStudent stud;
    stud.readPStudent("Ivan Ivanov", "8206123422", 42444,
                      6.0, 1000);
    stud.printPStudent();
    return 0;
}
```

Наследяване и достъп до наследените компоненти …

Достъп до методи чрез обекти на основния и производния клас

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

Обект на производен клас има пряк достъп до public компонентите на собствения си и компонентите на основния клас, наследени в производния клас като public. Последното е възможно, ако атрибутът за област на производния клас е public и компонентата е в public секция на основния клас.

Наследяване и достъп до наследените компоненти …

```
Person pe;
Student stud;
PStudent pstud;
```

Ще напомним, че Student е производен клас на основен клас Person с атрибут за област public, а PStudent е производен клас на основния клас Student също с атрибут за област public. Обектът pe има пряк достъп до public методите на Person, stud има пряк достъп до public методите на Person и Student, а pstud има пряк достъп до public методите на Person, Student и PStudent, т.е. допустими са обръщенията:

Наследяване и достъп до наследените компоненти …

```
pe.readPerson("Ivan Ivanov", "5804134986");
pe.printPerson();
stud.readStudent("Pavel Dimov", "4806193046", 30100, 4.50);
stud.readPerson("Pavel Dimov", "4806193046");
stud.printPerson();
stud.printStudent();
pstud.readPStudent("Pavel Dimov", "4806193046", 30100, 4.50, 500);
pstud.readStudent("Pavel Dimov", "4806193046", 30100, 4.50);
pstud.readPerson("Pavel Dimov", "4806193046");
pstud.printPerson();
pstud.printStudent();
pstud.printPStudent();
```

Наследяване и достъп до наследените компоненти …

Да се върнем към предишната програма. В нея класът Person е основен на класа Student с атрибут за област private. Student наследява всички секции на Person като private. Следователно обектите на Student нямат пряк достъп до методите на Person. Ако имаме дефинициите:

```
Person pe;
Student stud;
```

допустими са обръщенията

```
pe.readPerson("Ivan Ivanov", "5804134986");
pe.printPerson();
stud.readStudent("Pavel Dimov", "4806193046", 30100, 4.50);
stud.printStudent();
```

а обръщенията:

~~`stud.readPerson("Pavel Dimov", "4806193046");`~~

~~`stud.printStudent();`~~

са недопустими.

Наследяване и достъп до наследените компоненти …

Достъп на основен клас до членове на производен клас и обратно

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

Производният клас също няма привилигирован достъп до членове на основния клас. Производните класове нямат достъп до методите, обявени като private в основния клас.

Допустими са редица присвоявания между обекти на основния и производния клас. Ще ги разгледаме подробно по-нататък. Засега ще отбележим само, че за реализирането им се извършват редица преобразувания.

Наследяване и достъп до наследените компоненти …

Достъп на функции приятели на производен клас до компоненти на основния му клас

Ще напомним, че функциите-приятели на клас не са елементи на класа, на който са приятели. Те са външни функции, получили привилигирован достъп до компонентите на класа.

Функциите приятели на производен клас имат същите права на достъп като член-функциите на производния клас. *Имат пряк достъп до всички компоненти, декларирани в класа и до public и protected компонентите на основния клас. Декларацията за приятелство не се наследява. Функция приятел на базовия клас не е приятел (освен ако не е декларирана като такава) на производния клас*.

Наследяване и достъп до наследените компоненти …

Забележки:

1. Използването на секция protected позволява пряк достъп на производния клас до нейните компоненти. По такъв начин се нарушава принципът на капсулиране на данните, но пък дадената “привилегия” повишава ефективността на генерирания код.

2. Дефинирането на производни класове с атрибут за област private предизвиква забрана на достъпа на обект на класа до интерфейса на базовия му клас. Това се прави когато не трябва да се използва интерфейса на базовия клас, а се налага да се преработи и всички негови полезни функции да бъдат предефинирани.