Наследяване. Конструктори, деструктор

гл.ас., д-р. Нора Ангелова

НАСЛЕДЯВАНЕ

НАСЛЕДЯВАНЕ

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

НАСЛЕДЯВАНЕ

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, се нарича базов или основен клас, а новосъздаденият клас – производен.

НАСЛЕДЯВАНЕ

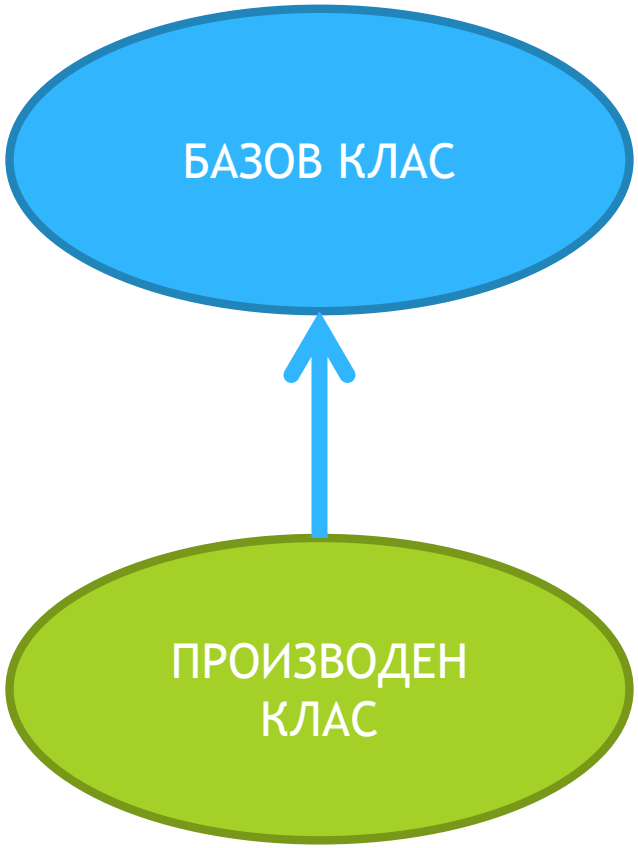
При създаване на нов клас, който има общи компоненти и поведение с вече дефиниран клас, вместо да дефинира повторно тези компоненти и поведение, програмистът може да определи новия клас като клас наследник на вече дефинирания.

- Базов и Производен клас

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

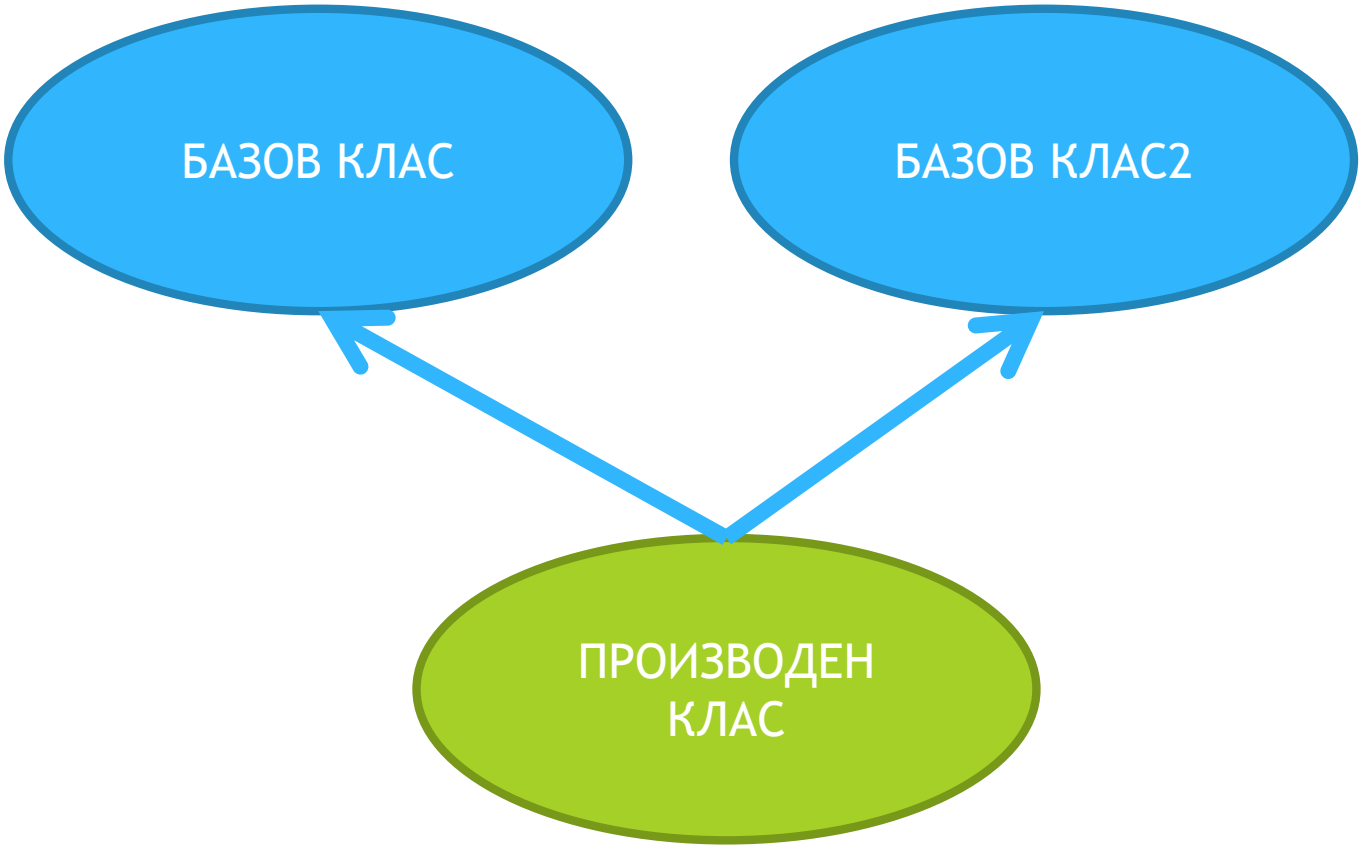
НАСЛЕДЯВАНЕ

- Единично наследавяне



НАСЛЕДЯВАНЕ

- Множествено наследяване



НАСЛЕДЯВАНЕ

- Множеството от компонентите на производен клас се състои от компонентите на неговите базови класове и компонентите, декларирани в самия производен клас.

- Дефиниция - Механизмът, чрез който производният клас получава компонентите на базовия, се нарича наследяване

НАСЛЕДЯВАНЕ

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.

НАСЛЕДЯВАНЕ

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**.

НАСЛЕДЯВАНЕ

- Дефиниция на производен клас

<заглавие> ::=
class <име_на_производен_клас> :
[<атрибут_за_област>] <име_на_базов_клас>
{ , [<атрибут_за_област>] <име_на_базов_клас> }

<име_на_производен_клас> ::= <идентификатор>
<атрибут_за_област> ::= public | protected | private
<име_на_базов_клас> ::= <идентификатор>

Наследяване

- Дефиниция на производен клас

Пример:

- Единично наследяване

```
class Point3D: public Point {
};
```

- Множествено наследяване

```
class CarPassport: private Car, protected Passport {
};
```

НАСЛЕДЯВАНЕ

- Дефиниция на производен клас

Атрибутът за достъп по подразбиране е private.

Пример:

```
class CarPassport: Car, Passport {
};
```

```
class CarPassport: private Car, private Passport {
};
```

НАСЛЕДЯВАНЕ

- Дефиниция на производен клас

Пример:

```
class CarPassport: public Car, Passport {
};
```

```
class CarPassport: public Car, private Passport {
};
```

НАСЛЕДЯВАНЕ

- Дефиниция на производен клас

Пример:

```
class CarPassport: Car, public Passport {
};
```

```
class CarPassport: private Car, public Passport {
};
```

НАСЛЕДЯВАНЕ

- Директни основни класове на производен клас. Директните основни класове се изброяват в заглавието на производния клас, предшествани от двоеточие (:).

Пример:

Класът People от предходната задача е директен основен клас на класа Student.

- Индиректни основни класове - не се изброяват в заглавието на производните класове, но се наследяват от две или повече по-високи нива

НАСЛЕДЯВАНЕ

- Достъп до наследените компоненти

Атрибут public

public		public
private		private
protected		protected

Атрибут private

public		private
private		private
protected		private

НАСЛЕДЯВАНЕ

- Достъп до наследените компоненти

Атрибут protected

public		protected
private		private
protected		protected

НАСЛЕДЯВАНЕ

- Достъп до наследените компоненти

Атрибут public

public		public
private		private
protected		protected

Пример:

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

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

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

НАСЛЕДЯВАНЕ

- Достъп до наследените компоненти

Атрибут private

public		private
private		private
protected		private

Пример:

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

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

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

НАСЛЕДЯВАНЕ

- Достъп до наследените компоненти

Атрибут protected

public		protected
private		private
protected		protected

Пример:

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

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

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

НАСЛЕДЯВАНЕ

- Достъп до наследените компоненти

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

Както при класовете без наследяване,
собствените компоненти на производния клас имат пряк достъп помежду си.

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

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

НАСЛЕДЯВАНЕ

- Пряк достъп (ПД), на член-функции на производен клас до компонентите на базовия му клас
  `class DerivativeClassName { … // ПД до BaseClass }`
  и
  външен достъп (ВД) на обект на производен клас до компонентите на базовия му клас.
  `class DerivativeClassName { … // ПД до BaseClass}`
  `DerivativeClassName obj; // ВД до BaseClass obj.`

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

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 (освен ако не е декларирана като такава) на производния му клас

Предефиниране на компоненти

- Проблем - производният клас може да наследи член-функция, която не трябва да има.
- Решение – осъществява се като в производния клас се предефинира (дефинира се повторно) член-функцията с подходяща реализация.

ПРЕДЕФИНИРАНЕ НА КОМПОНЕНТИ

- Проблем - базовият и производният 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.

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

  <име_на_основен_клас>::<компонента>

ПРЕДЕФИНИРАНЕ НА КОМПОНЕНТИ

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


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

```
int main() {
  base b;
  der d;
  b.init(5);    // base::init
  d.init(10);   // der::init
  b.display(); // base::display
  d.display(); // der::display
  d.base::init(20); // base::init
  d.base::display();
                    // base::display

  return 0;
}
```

Единично наследяване. Конструктори, деструктор и операторна функция за присвояване на производен клас

ЗАБЕЛЕЖКА

- Обикновените конструктори, конструкторът за присвояване, операторната функция за присвояване и деструкторът са методи, за които не важат правилата за достъп при наследяване.
- Тези методи на основния клас (**с някой изключения**) не се наследяват от производния клас.

КОНСТРУКТОР

- Конструкторите на производния клас инициализират само собствените член-данни на класа.
- Наследените член-данни на производния клас се инициализират от конструктор на основния му клас.

  Реализация:
  Това се осъществява като в дефиницията на конструктора на производния клас се укаже обръщение към съответен конструктор на основния клас.

КОНСТРУКТОР

<дефиниция_на_конструктор_на_производен_клас> ::=
<име_на_производен_клас>::<име_на_производен_клас>
(<параметри>) <инициализиращ_списък>
{
  <тяло>
}

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

ЗАБЕЛЕЖКА

- При единичното наследяване инициализиращият списък на конструктора на производния клас може да съдържа не повече от едно обръщение към конструктор на основен клас.

- Ако инициализиращият списък не съдържа обръщение към конструктор на основния клас, чрез което да укаже как да се инициализира наследената част, в базовия клас трябва да е дефиниран конструктор по подразбиране.

- Освен обръщение към конструктор на базовия клас, инициализиращият списък на конструктора на производния клас може да съдържа инициализация на собствени за производния клас член-данни

- Обръщението към конструктора на основния клас се записва в дефиницията на конструктора на производния клас, а не в неговата декларация в тялото на производния клас.

ПРИМЕР

```
class base {
protected:
  int a2;
private:
  int a1;
public:
  // конструктор по подразбиране
  base() {
    a1 = 0;
    a2 = 0;
  }
  // конструктор с един параметър
  base(int x) {
    a1 = x;
  }
  // конструктор с 2 параметъра
  base(int x, int y) {
    a1 = x;
    a2 = y;
  }
  void a3() const {
    cout << "a1: " << a1 << endl
         << "a2: " << a2 << endl;
  }
};
```

```
class der : public base {
protected:
  int d2;
private:
  int d1;
public:
  der(int x, int y, int z, int t) : base(x, y) {
    d1 = z;
    d2 = t;
  }

  void d3() const {
    cout << "d1: " << d1 << endl
         << "d2: " << d2 << endl
         << "a2: " << a2 << endl;
    cout << "a3():" << endl;
    a3();
  }
};
```

```
der x(1, 2, 3, 4);
x.d3();
```

Резултат:

```
d1: 3
d2: 4
a2: 2
a3():
a1: 1
a2: 2
```

ПРИМЕР

```
class base {
protected:
  int a2;
private:
  int a1;
public:
  // конструктор по подразбиране
  base() {
    a1 = 0;
    a2 = 0;
  }
  // конструктор с един параметър
  base(int x) {
    a1 = x;
  }
  // конструктор с 2 параметъра
  base(int x, int y) {
    a1 = x;
    a2 = y;
  }
  void a3() const {
    cout << "a1: " << a1 << endl
         << "a2: " << a2 << endl;
  }
};
```

```
class der : public base {
protected:
  int d2;
private:
  int d1;
public:
  der(int x, int y, int z, int t) : base() {
    d1 = z;
    d2 = t;
  }
```

Наследените компоненти се инициализират от подразбиращия се конструктор

```
  void d3() const {
    cout << "d1: " << d1 << endl
         << "d2: " << d2 << endl
         << "a2: " << a2 << endl;
    cout << "a3():" << endl;
    a3();
  }
};
```

ПРИМЕР

```
class base {
protected:
  int a2;
private:
  int a1;
public:
  // конструктор по подразбиране
  base() {
    a1 = 0;
    a2 = 0;
  }
  // конструктор с един параметър
  base(int x) {
    a1 = x;
  }
  // конструктор с 2 параметъра
  base(int x, int y) {
    a1 = x;
    a2 = y;
  }
  void a3() const {
    cout << "a1: " << a1 << endl
         << "a2: " << a2 << endl;
  }
};
```

```
class der : public base {
protected:
  int d2;
private:
  int d1;
public:
  der(int x, int y, int z, int t) {
    d1 = z;
    d2 = t;
  }
```

Наследените компоненти се инициализират от подразбиращия се конструктор

```
  void d3() const {
    cout << "d1: " << d1 << endl
         << "d2: " << d2 << endl
         << "a2: " << a2 << endl;
    cout << "a3():" << endl;
    a3();
  }
};
```

ПРИМЕР

```
class base {
protected:
  int a2;
private:
  int a1;
public:
  // конструктор по подразбиране
  base() {
    a1 = 0;
    a2 = 0;
  }
  // конструктор с един параметър
  base(int x) {
    a1 = x;
  }
  // конструктор с 2 параметъра
  base(int x, int y) {
    a1 = x;
    a2 = y;
  }
  void a3() const {
    cout << "a1: " << a1 << endl
         << "a2: " << a2 << endl;
  }
};
```

```
class der : public base {
protected:
  int d2;
private:
  int d1;
public:
  der(int x, int y, int z, int t): base(), base(x, y)
  {
    d1 = z;
    d2 = t;
  }
```

ГРЕШКА - не повече от 1 извикване към конструктор на базов клас

```
  void d3() const {
    cout << "d1: " << d1 << endl
         << "d2: " << d2 << endl
         << "a2: " << a2 << endl;
    cout << "a3():" << endl;
    a3();
  }
};
```

ЗАБЕЛЕЖКА

- Ако производният клас има собствени член-данни, които са обекти на класове и в инициализиращия списък на конструктора не е указано как те да се инициализират, техните конструктори по подразбиране се извикват след изпълнението на обръщението към конструктора на основния клас от инициализиращия списък и преди изпълнението на операторите в тялото на конструктора на производния клас.
- Редът на изпълнението им съвпада с реда на член-данните обекти в производния клас

ПРИМЕР

```
class base {
protected: int a2;
private: int a1;
public:
  base() {
    cout << "constructor base() \n";
    a1 = a2 = 0;
  }

  base(int x, int y) {
    cout << "constructor base("
         << x << ", " << y << ")\n";
    a1 = x;
    a2 = y;
  }

  void a3() const {
    cout << "a1: " << a1 << endl
         << "a2: " << a2 << endl;
  }
};
```

```
class der : public base {
protected: base d2;
private: base d1;
public:
  der(int x, int y) : base(x, y) {
    cout << "constructor der\n";
  }

  void d3() const {
    d1.a3();
    d2.a3();
    cout << "a2: " << a2 << endl;
    cout << "a3():" << endl;
    a3();
  }
};
```

```
int main() {
  der x(1, 2);
  x.d3();
  return 0;
}
```

```
constrictor base(1, 2)
constrictor base()
constrictor base()
constrictor der
a1: 0
a2: 0
a1: 0
a2: 0
a2: 2
a3():
a1: 1
a2: 2
```

ПРИМЕР

```
class base {
protected: int a2;
private: int a1;
public:
  base() {
    cout << "constructor base() \n";
    a1 = a2 = 0;
  }

  base(int x, int y) {
    cout << "constructor base("
         << x << ", " << y << ")\n";
    a1 = x;
    a2 = y;
  }

  void a3() const {
    cout << "a1: " << a1 << endl
         << "a2: " << a2 << endl;
  }
};
```

```
class der : public base {
protected: base d2;
private: base d1;
public:
  der(int x, int y) : base(x, y) {
    cout << "constructor der\n";
    d1 = base(15, 25);
     d2 = base(35, 45);
  }

  void d3() const {
    d1.a3();
    d2.a3();
    cout << "a2: " << a2 << endl;
    cout << "a3():" << endl;
    a3();
  }
};
```

```
int main() {
  der x(1, 2);
  x.d3();
  return 0;
}
```

```
constrictor base(1, 2)
constrictor base()
constrictor base()
constrictor der
constructor base(15, 25)
constructor base(35, 45)
a1: 15
a2: 25
a1: 35
a2: 45
a2: 2
a3():
a1: 1
a2: 2
```

ПРИМЕР

```
class base {
protected: int a2;
private: int a1;
public:
  base() {
    cout << "constructor base() \n";
    a1 = a2 = 0;
  }

  base(int x, int y) {
    cout << "constructor base("
         << x << ", " << y << ")\n";
    a1 = x;
    a2 = y;
  }

  void a3() const {
    cout << "a1: " << a1 << endl
         << "a2: " << a2 << endl;
  }
};
```

```
class der : public base {
protected: base d2;
private: base d1;
public:
//Избягва се двукратното инициализиране
  der(int x, int y) : base(x, y),
  d1(15, 25), d2(35, 45) {
    cout << "constructor der\n";
  }

  void d3() const {
    d1.a3();
    d2.a3();
    cout << "a2: " << a2 << endl;
    cout << "a3():" << endl;
    a3();
  }
};
```

```
int main() {
  der x(1, 2);
  x.d3();
  return 0;
}
```

```
constrictor base(1, 2)
constructor base(35, 45)
constructor base(15, 25)
constrictor der
a1: 15
a2: 25
a1: 35
a2: 45
a2: 2
a3():
a1: 1
a2: 2
```

Случаи

- В основния клас не е дефиниран конструктор в т.ч. конструктор за присвояване

В този случай в инициализиращия списък на конструктор(ите) на производния клас не трябва да се задава инициализация за наследените от основния клас член-данни. Наследената част на производния клас остава неинициализирана.

Случаи

- В основния клас е дефиниран само един конструктор с параметри, който не е подразбиращият се

Възможни са:

а) в производния клас е дефиниран конструктор

В този случай в инициализиращия списък на конструктора на производния клас задължително трябва да има обръщение към конструктора с параметри на основния клас. Изпълнява се по начина, описан по-горе.

б) в производния клас не е дефиниран конструктор

В този случай компилаторът ще сигнализира за грешка. Необходимо е да се създаде конструктор на производния клас, който да извика конструктора на основния клас

СЛУЧАИ

- В основния клас са дефинирани няколко конструктора в т.ч. подразбиращ се конструктор

Възможни са:

а) в производния клас е дефиниран конструктор

Тогава в инициализиращия списък на конструктора на производния клас може да се посочи, но може и да не се посочи конструктор на основния клас. Ако не е посочен, компилаторът се обръща към конструктора по подразбиране на основния клас.

б) в производния клас не е дефиниран конструктор

В този случай компилаторът автоматично създава конструктор по подразбиране за производния клас. Последният активира и изпълнява конструктора по подразбиране на основния клас.

Собствените член-данни на производния клас остават неопределени.

ДЕСТРУКТОР

- Деструкторът на производен клас трябва да разруши само онези собствени на производния клас член-данни, които са разположени в динамичната памет.

  Деструкторите на производен клас и на неговия основен клас се изпълняват автоматично в ред, обратен на реда на изпълнението на техните конструктори.

  Най-напред се изпълнява деструкторът на производния клас, след това се изпълнява деструкторът на основния му клас.

КРАЙ