Dziedziczenie (tematy zaawansowane)

Ogólna postać dziedziczenia klas:

class nazwa_clasy_pochodnej : *specyfikator_dostępu* nazwa_clasy_bazowej
{

};

specyfikator_dostępu : public private protected

- Specyfikator dostępu definiuje, w jaki sposób składowe klasy bazowej dziedziczą się klasą pochodną. Jeśli specyfikator dostępu nie został określony, domyślnie jest przyjęte: dla klasy pochodnej, zadeklarowanej z słowem kluczowym *class*, jako *private*; z słowem kluczowym *struct* – jako *public*.

- *public*:
  - składowe publiczne klasy bazowej stają publicznymi składowymi klasy pochodnej (bezpośrednio dostępnymi wewnątrz i zewnątrz klasy pochodnej).
  - składowe prywatne klasy bazowej pozostają prywatnymi składowymi klasy pochodnej. To oznacza, że dostęp do tych składowych jest możliwy tylko przez funkcje typu get_val() jak w klasie pochodnej, tak i zewnątrz klasy pochodnej.

- składowe chronione (protected) klasy bazowej stają składowymi chronionymi klasy pochodnej - bezpośrednio dostępnymi składowymi w klasie pochodnej, przecież pozostają ukrytymi zewnątrz klasy pochodnej i klasy bazowej.

- *private*:
  - składowe publiczne klasy bazowej stają prywatnymi składowymi klasy pochodnej. Wewnątrz klasy pochodnej bezpośredni dostęp do tych składowych istnieje, zewnątrz – nie istnieje. Klasa pochodna może udostępnić publiczne składowe klasy bazowej przez swoi metody *get_derived*() lub przewrócić ich. (patrz slajd 4).
  - składowe prywatne klasy bazowej pozostają prywatnymi składowymi klasy pochodnej. Klasa pochodna nie ma dostępu bezpośredniego do tych składowych (dostęp tylko przez publiczne metody klasy bazowej), zewnątrz dostęp do ukrytych składowych klasy bazowej przez metody klasy bazowej jest niemożliwy (publiczne metody klasy bazowej stają składowymi ukrytymi klasy pochodnej).
  - składowe chronione (*protected*) klasy bazowej stają składowymi prywatnymi klasy pochodnej - są dostępne bezpośrednio w klasie pochodnej i nie są dostępne na zewnątrz klasy pochodnej.

- *protected*:
  - składowe publiczne klasy bazowej stają składowymi chronionymi klasy pochodnej;
  - składowe prywatne klasy bazowej pozostają składowymi prywatnymi klasy pochodnej;
  - składowe chronione (*protected*) klasy bazowej pozostają składowymi chronionymi klasy pochodnej.
- Przewrócenie składowych publicznych klasy bazowej przy dziedziczeni jako *private*:

```
class B
{
        int i;
public:
        int j;
.....
};

class D : private B
{
public:
        B::j;     //przewrócenie publicznej składowej klasy bazowej
                  //do statusu public przy dziedziczeni jako private.
        B::i;     //błąd! i – składowa ukryta klasy bazowej
};

int main()
{
    D ob;
    ob.j = 20; //OK
    return 0;
}
```

Specyfikator dostępu składowej klasy bazowej	Klasa pochodna dziedziczy się jako		
	: public	: private	: protected
public: (składowa jest dostępna zewnątrz klasy bazowej – ob.i = … // OK)	**wewnątrz klasy pochodnej**: jest dostępna	**wewnątrz klasy pochodnej**: jest dostępna	**wewnątrz klasy pochodnej**: jest dostępna
	zewnątrz klasy pochodnej: jest dostępna	**zewnątrz klasy pochodnej:** nie jest dostępna	**zewnątrz klasy pochodnej:** nie jest dostępna
private: (składowa nie jest dostępna zewnątrz klasy bazowej – ob.i = … // !OK)	**wewnątrz klasy pochodnej**: nie jest dostępna	**wewnątrz klasy pochodnej**: nie jest dostępna	**wewnątrz klasy pochodnej**: nie jest dostępna
	zewnątrz klasy pochodnej: nie jest dostępna	**zewnątrz klasy pochodnej:** nie jest dostępna	**zewnątrz klasy pochodnej:** nie jest dostępna
protected: (składowa nie jest dostępna zewnątrz klasy bazowej – ob.i = … // !OK)	**wewnątrz klasy pochodnej**: jest dostępna	**wewnątrz klasy pochodnej**: jest dostępna	**wewnątrz klasy pochodnej**: jest dostępna
	zewnątrz klasy pochodnej: nie jest dostępna	**zewnątrz klasy pochodnej:** nie jest dostępna	**zewnątrz klasy pochodnej:** nie jest dostępna

- Specyfikatory określające sposób dziedziczenia klasy bazowej przez klasę pochodną nie wpływają na sposób uzyskiwania dostępu do elementów klasy bazowej w klasie pochodnej. Dostęp do elementów klasy bazowej w klasie pochodnej jest określany tylko przez specyfikator dostępu, który te elementy są zadeklarowane w klasie bazowej.

Konstruktory, destruktory i dziedziczenie

- Jeśli klasa bazowa i klasa pochodna mają konstruktory i destruktory, to przy tworzeniu obiektu klasy pochodnej kolejność wywołani jest taka:
  - konstruktor klasy bazowej
  - konstruktor klasy pochodnej
  - destruktor klasy pochodnej
  - destruktor klasy bazowej

- Klasa bazowa nic nie wie o istnieniu klasy pochodnej – inicjowanie w klasie bazowej jest wykonywane niezależnie od klasy pochodnej. Inicjowanie klasy bazowej może być podstawą dla inicjowania klasy pochodnej. Dla tego konstruktor klasy bazowej jest wywołany wcześniej od konstruktora klasy pochodnej.

- Przy niszczeniu obiektów niszczenie składowych klasy bazowej powodowało by uszkodzenie obiektu klasy pochodnej. Dla tego destruktor klasy pochodnej ma być wywołany wcześniej od destruktora klasy bazowej.

- Przekazanie argumentów konstruktorowi klasy bazowej:
  - wszystkie argumenty klasy pochodnej, a również, klasy bazowej, pozostają przekazane konstruktorowi klasy pochodnej.
  - argumenty klasy bazowej pozostają przekazane do klasy bazowej:

```
konstr_klasy_pochod(lista_argum) : konstr_klasy_bazowej(lista_argum_baz)
{
    //ciało konstruktora klasy pochodnej
}
```

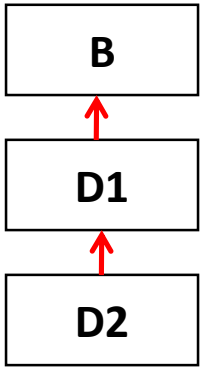
lista_argum – lista argumentów klasy pochodnej i klasy bazowej (deklaracja razem z ich typami)
lista_argum_baz – lista argumentów klasy bazowej (przekazywanie jako argumentów faktycznych)

- **Example 37.**

- Przykład:

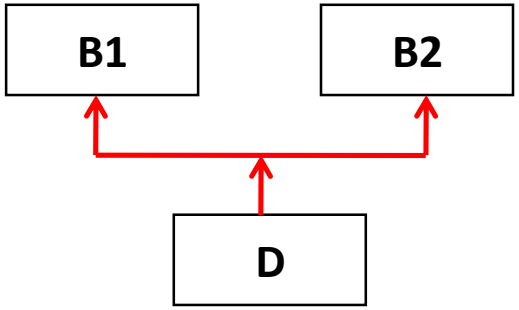
```
class base
{
        int i;
public:
        base(int ii) { i = ii;}
};

class derived : public base
{
        int j;
public:
        derived(int ii) : base(ii) { j=0; }      //przekazanie argumentu do klasy bazowej
                                                 //klasa pochodna argumentów nie potrzebuje
};
```

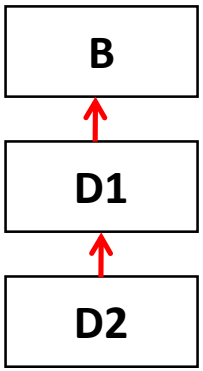
Wielodziedziczenie.



Wielopoziomowa hierarchia klas

Klasa pochodna dziedziczy różne klasy bazowe



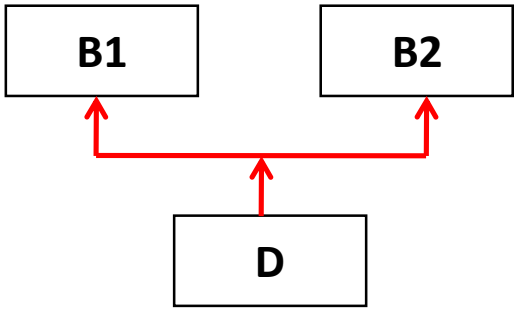


➢ Przy hierarchii klas wielopoziomowej konstruktory się wywołują w tej kolejności, w którą te klasy dziedziczą, a destruktory – w odwrotnej kolejności:

- constructor B
- constructor D1
- constructor D2
- destructor D2
- destructor D1
- destructor B

Definicje klas pochodnych:

```
class D1 : public B
 {
 };

 class D2 : public D1
 {
 };
```



➢ Jeśli klasa pochodna dziedziczy kilka klas bazowych, deklaracja klasy pochodnej jest taka:

```
class D : public B1, public B2
{
};

class derived_class_name : specyf_dostępu  base_class_name_1,
                           specyf_dostępu  base_class_name_2,
                           ……..
                           specyf_dostępu  base_class_name_N
{
};
```

➢ Konstruktory są wywołane w kolejności tworzenia klas, w porządku od lewej do prawej, czyli tak, jak klasy bazowe zostały wymienione na liście dziedziczenia. Destruktory są wywołane w odwrotnej kolejności, od prawej do lewej:

```
constructor B1
constructor B2
constructor D
destructor D
destructor B2
destructor B1
```

➢ A w takim przypadku konstruktory i destruktory będą wywołane tak:

```
class  D : private B2 , private B1
{
}

constructor B2
constructor B1
constructor D
destructor D
destructor B1
destructor B2
```

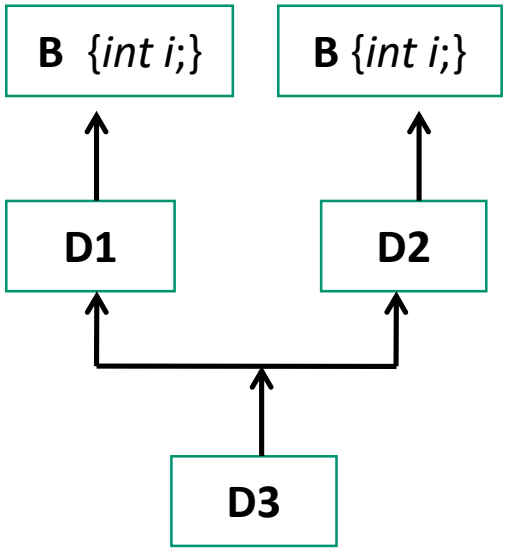
- Przekazywanie argumentów konstruktorom klas bazowych.

```
konstr_klasy_pochod(lista_argum) : nazwa_klasy_bazow_1(lista_arg_1) ,
                                   nazwa_klasy_bazow_2(lista_arg_2) ,
                                   ……,
                                   nazwa_klasy_bazow_N(lista_arg_N)
{
      // konstruktor klasy pochodnej
}
```

lista_argum – ogólna lista argumentów dla klasy pochodnej i wszystkich klas bazowych
lista_argum_1 – lista argumentów dla klasy bazowej 1,
lista_argum_2 – lista argumentów dla klasy bazowej 2,
…….
lista_argum_N – lista argumentów dla klasy bazowej N

Przykłady W38, W39

- Rozważymy taką sytuacje:



- Przy takiej hierarchii klas powstaje niejednoznaczność. Niech klasa B ma składową *int i*. Wtedy klasy D1, D2 (pochodne od klasy B, więc każda z tych klas zawiera swoją własną kopie klasy bazowej) będą mieli dwa rożne egzemplarze składowej *i*, które się znajdują w różnych adresach pamięci. Klasa D3 dziedziczy klasy D1, D2. Więc w klasie D3 powstają 2 egzemplarze składowej *i*. Który egzemplarz składowej *i* powinni ż my pobrać?

- Klasy wirtualne zapobiegają powstaniu dwóch kopii w takiej sytuacji:

```
class D1 : virtual public B
{
}

class D2 : virtual public B
{
}

class D3 : public D1, public D2
{
}
```

Przykład W40.

- Jeśli klasa bazowa B została oddziedziczona przez D1, D2 jako klasa wirtualna, jedna kopia tej klasy bazowej powstaje wewnątrz klasy pochodnej.

- Różnica pomiędzy klasą zwykłą i wirtualną pojawia się wtedy, gdy obiekt dziedziczy klasą bazową więcej niż jeden raz.

- **Przykład W41 (from Schildt)) !**