

Wykład 8: Klasy cz. 3



Składniki statyczne klas

Składniki statyczne klas

Pola statyczne – stosujemy, gdy poszczególne egzemplarze obiektów danej klasy powinny posługiwać się tą samą daną.

Dana statyczna jest w pamięci tworzona jednokrotnie i jest wspólna dla wszystkich egzemplarzy obiektów danej klasy. Co więcej: istnieje nawet wtedy, gdy jeszcze nie zdefiniowaliśmy ani jednego egzemplarza obiektu tej klasy.

```
class klasa{  
    public :  
    int x ;  
    static int skladnik ;  
} ;
```



- ✓ Deklaracja składnika statycznego w ciele klasy nie jest jego definicją.
- ✓ Definicję musimy umieścić gdzieś tak, by miała zakres pliku. Czyli tak, jakbyśmy umieszczali definicję zmiennej globalnej.
- ✓ Definicja taka może zawierać inicjalizację.

```
int klasa::składnik = 6 ;
```

- ✓ Składnik statyczny może być także typu private. Inicjalizacja składnika statycznego możliwa jest nawet jeśli jest on typu private. Po inicjalizacji prywatny składnik statyczny nie może być przez obcych czytany ani zapisywany.

Składniki statyczne klas

Do składnika statycznego można odwołać się na trzy sposoby:

- ✓ Za pomocą nazwy klasy i operatora zakresu „ :: ”

klasa :: składnik

- ✓ Jeśli istnieją już jakieś egzemplarze obiektów klasy, to możemy posłużyć się operatorem „ . ”

obiekt . składnik

- ✓ Jeśli mamy wskaźnik do obiektu stosujemy operator „ -> ”

***wsk = &obiekt;**

wsk->składnik;

Składniki statyczne klas

```
1  #include <iostream>
2  class Klasa
3  {
4      static int wspolne;
5  public:
6      void metoda() {
7          std::cout << wspolne << std::endl;
8          ++wspolne;
9      }
10 };
11
12 int Klasa::wspolne = 0;
13
14 int main()
15 {
16     Klasa a, b, c;
17 }
```



Obiekty i funkcje const



Funkcje składowe typu const – funkcje takie nie mogą modyfikować składników klas.


```
1  #include <iostream>
2
3  using namespace std;
4
5  class pozycja {
6      int x , y ;
7      public :
8          pozycja(int a, int b) {x = a; y = b;}
9          void wypis (void) const ;
10         void przesun (int a, int b);
11     } ;
12     void pozycja::wypis()    const {
13         cout << x << " , " << y << endl ;
14     }
15
16     void pozycja::przesun(int a , int b) {
17         x = a; y = b ;
18     }
19     main() {
20         pozycja samochód(40,50);
21         const pozycja dom(50,50);
22         samochód.wypis() ;
23         dom.wypis() ;
24         samochód.przesun(4,10);
25         // dom.przesun(0, 0); //błąd!
26     }
```



Funkcje zaprzyjaźnione

Funkcje zaprzyjaźnione

Funkcja zaprzyjaźniona – to funkcja która ma prawo dostępu do prywatnych składników klasy.

- ✓ Wewnątrz definicji klasy wystarczy umieścić deklarację tej funkcji poprzedzoną słowem **friend**.
- ✓ Uwaga: to nie funkcja ma twierdzić, że jest zaprzyjaźniona. To klasa ma zadeklarować, że przyjaźni się z tą funkcją i nadaje jej prawo dostępu do składników prywatnych. Zatem słowo **friend** pojawia się tylko wewnątrz definicji klasy.

Funkcje zaprzyjaźnione

```
1  class punkt {  
2      int x,y ;  
3      friend void gdzie_jest (punkt &) ;  
4  } ;  
5  
6  void gdzie_jest (punkt & p)  
7  {  
8      cout << <<" (" << p.x << " ; " << p.y <<")";  
9  }  
10
```



Konstruktor i destruktor c.d.

Konstruktor i destruktor

Jawne wywołanie konstruktora

Obiekt może być też stworzony przez jawne wywołanie konstruktora.

W efekcie otrzymujemy obiekt, który nie ma nazwy, a czas jego życia ogranicza się do wyrażenia, w którym go użyto.

`nazwa_klasy(argumenty)`

Uwaga: nie stosujemy zapisu:

~~`obiekt.funkecja_skladowa(argumenty)`~~

Konstruktor nie jest wywoływany na rzecz jakiegoś obiektu, bo ten obiekt jeszcze nie istnieje. Zadaniem konstruktora jest go utworzyć.



Jawne wywołanie destruktor

Należy podać całą jego nazwę. Jawne wywołanie destruktor nie może się zacząć od ~(wężyka) i wcześniej musi być albo obiekt, na rzecz którego jest wywoływany i kropka lub wskaźnik do obiektu „->”

```
obiekt.~klasa();  
wskaznik->~klasa();
```

Konstruktor i destruktor

Konstruktor domniemany

Konstruktor domniemany to taki konstruktor, który można wywołać bez żadnego argumentu.

```
class klasa {  
    public :  
        klasa(void) ; //konst.    domniemany  
};
```




Przeciążanie konstruktorów

Jedna klasa może posiadać kilka konstruktorów

```
class klasa {  
    public :  
        klasa(void) ; //konst.   domniemany  
        klasa(int) ;  
        klasa(float)  
        klasa(int, int) ;  
        // klasa (int a=5, char *s = NULL) ;  
        //   to też może być konst. Domniemany ale tylko keden w klasie  
};
```



Lista inicjalizacyjna konstruktora

```
class abc {  
    const abc(): float x, int a, char c  
    {}  
  
// deklaracja konstruktora  
abc(float pp, int dd, char znak);
```



Dziedziczenie



Dziedziczenie - pozwala klasie odziedziczyć zmienne i metody po drugiej. Klasę dziedziczącą nazywamy **klasą pochodną**, a klasę, po której klasa pochodna dziedziczy, nazywamy **klasą bazową**. (Klasa pochodna pochodzi od bazowej).

```
class Bazowa  
{  
};
```

```
class Pochodna : typ_dziedziczenia Bazowa  
{  
};
```



W klasie pochodnej możemy:

- **zdefiniować dodatkowe dane składowe,**
- **zdefiniować dodatkowe funkcje składowe, zdefiniować składnik (najczęściej funkcję składową), który istnieje już w klasie podstawowej.**

Definiowanie nowych funkcji składowych - bez definiowania dodatkowych danych składowych także ma sens. jest to jakby wyposażenie klasy w nowe zachowania;



Typy dziedziczenia:

- 1. Dziedziczenie publiczne** - najczęściej stosowane. Składowe publiczne klasy bazowej są odziedziczone jako publiczne, a składowe chronione jako chronione.
- 2. Dziedziczenie chronione** - składowe publiczne są dziedziczone jako chronione, a składowe chronione jako chronione.
- 3. Dziedziczenie prywatne** - jest domyślne (gdy nie jest podany typ dziedziczenia). Składowe publiczne są dziedziczone jako prywatne, a chronione jako prywatne.



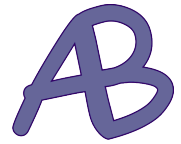
Typy dziedziczenia:

```

/*
+-----+-----+-----+-----+
|                sposób dziedziczenia                |
+-----+-----+-----+-----+
| public  | protected | private |
+-----+-----+-----+-----+
|          | public  # public  | protected | private |
| widoczność w klasie bazowej | protected # protected | protected | private |
|          | private #    -*   |    -*   |    -*   |
+-----+-----+-----+-----+

```

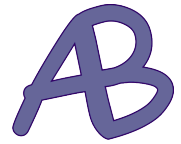
* - niedostępne, jeśli nie ma przyjaźni



Inicjalizacja odziedziczonych składowych

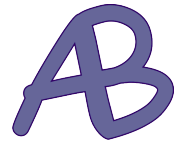
By zainicjalizować odziedziczoną część obiektu, wywołujemy w konstruktorze klasy pochodnej konstruktor klasy bazowej

```
1  class Bazowa
2  {
3      int x;
4  public:
5      Bazowa( int a ): x( a )
6      { }
7  };
8
9  class Pochodna: public Bazowa
10 {
11 public:
12     Pochodna( int a ):Bazowa( a ) //Jawne wywołanie konstruktora
13     { }
14 }
```

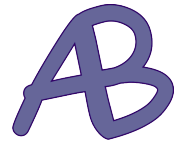
Dziedziczenie kilkupokoleniowe

```
class A {  
    // - ciało klasy A  
};  
class B :public A {  
    // - ciało klasy B  
};  
class C :public B {  
    // - ciało klasy C  
};
```



Konstruktor klasy pochodnej

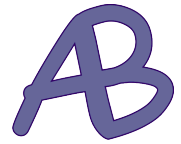
```
1  class A { public:  
2      A(); // konstr domniemany  
3      A(float); // inny konstruktor  
4  };  
5  
6  class B: public A { public:  
7      B(); // deklaracja konstruktora  
8  };  
9  
10     B::B(int x) : A() //lista inicjalizacyjna z wywołaniem  
11         // konstruktora klasy podstawowej  
12 {  
13     //... ciało konstruktora  
14 }  
15
```



Jeśli na liście inicjalizacyjnej nie umieściliśmy wywołania konstruktora klasy podstawowej, a klasa podstawowa w zestawie swoich konstruktorów nie ma konstruktora domniemanego - to wówczas kompilator uzna to za błąd.

Na liście inicjalizacyjnej można pominąć wywołanie konstruktora bezpośredniej klasy podstawowej tylko wtedy, gdy:

- klasa podstawowa nie ma żadnego konstruktora.
- ma konstruktory, a wśród nich jest konstruktor domniemany.



Dziedziczenie wielokrotne - klasa może wywodzić się bezpośrednio od więcej niż jednej klasy. Takie dziedziczenie nazywamy dziedziczeniem wielokrotnym.

- 1) Dana klasa podstawowa może na liście pochodzenia pojawić się tylko raz. Zatem błędem byłoby powiedzenie, że amfibia pochodzi od samochodu, łódki i jeszcze raz od samochodu.
- 2) Definicja klasy umieszczonej na liście pochodzenia - musi być już znana kompilatorowi. Nie wystarcza deklaracja zapowiadająca (zwiastująca):

Literatura:

W prezentacji wykorzystano przykłady i fragmenty:

- Grębosz J.: ***Symfonia C++***, ***Programowanie w języku C++ orientowane obiektowo***, Wydawnictwo Edition 2000.
- Jakubczyk K.: *Turbo Pascal i Borland C++ Przykłady*, Helion.

Warto zająrzeć także do:

- Sokół R.: ***Microsoft Visual Studio 2012 Programowanie w Ci C++***, Helion.
- Kernighan B.W., Ritchie D. M.: ***język ANSI C***, Wydawnictwo Naukowo Techniczne.

Dla bardziej zaawansowanych:

- Grębosz J.: ***Pasja C++***, Wydawnictwo Edition 2000.
- Meyers S.: ***język C++ bardziej efektywnie***, Wydawnictwo Naukowo Techniczne