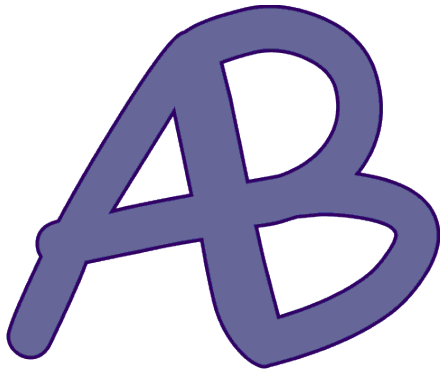


WYKŁAD: Programowanie obiektowe



dr Artur Bartoszewski
Katedra Informatyki
UTH Radom

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;



Klasa nie dziedziczy:

- ✓ Konstruktorów,
- ✓ Destrukтора,
- ✓ Operatora przypisania



Typy dziedziczenia



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
widoczność		public	protected	private
w klasie		protected	protected	private
bazowej		private	-*	-*

* - niedostępne, jeśli nie ma przyjaźni



Dziedziczenie a zasięg widoczności nazw

Przesłanianie dziedziczonych metod



Dziedziczenie a zasięg widoczności nazw

Przesłanianie składowych

Jeśli składowa klasy pochodnej ma taką samą nazwę jak składowa klasy bazowej - składowa klasy bazowej zostanie przesłonięta.

W klasie bazowej można użyć przesłoniętej składowej klasy bazowej - za pomocą operatora zasięgu ::

`KlasaBazowa :: pole;`

Przesłaniać możemy zarówno pola, jak i metody.



Uwaga: Przeciążanie metod działa tylko w obrębie klasy.

Jeśli metody wewnątrz jednej klasy mają taką samą nazwę, lecz różną listę parametrów mamy do czynienia z przeciążaniem metod klasy.

Jednak mechanizm nie działa w przypadku dziedziczenia – metoda klasy bazowej jest przesłaniana przez metodę klasy pochodnej o takiej samej nazwie



Dziedziczenie a zasięg widoczności nazw (przesłanianie składowych)

```
5  class bazowa
6  {
7  public:
8      int x=10;
9  };
10
11 class pochodna : public bazowa
12 {
13     int x=1000;
14 public:
15     void wypisz()
16     {
17         cout <<"x z klasy pochodnej = "<<x<<endl;;
18         cout <<"x z klasy bazowej = "<<bazowa::x<<endl;
19     }
20 };
21
22 int main()
23 {
24     pochodna k1;
25     k1.wypisz();
26     return 0;
27 }
```

```
x z klasy pochodnej = 1000
x z klasy bazowej = 10
```



Inicjalizacja odziedziczonych składowych

Konstruktor klasy pochodnej



Inicjalizacja odziedziczonych składowych

Aby 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 }
```



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 - przykład

```
5  class punkt
6  {
7  private:
8      int x;
9      int y;
10 public:
11     punkt (int px=0, int py=0)
12     {
13         x=px;
14         y=py;
15     }
16     void wypisz ()
17     {
18         cout <<"x = "<<x<<endl;
19         cout <<"y = "<<y<<endl;
20     }
21 };;
```


Dziedziczenie - przykład

```
23 class okrag : public punkt
24 {
25     private:
26         int r;
27     public:
28         okrag(int px=0, int py=0, int pr=0) : punkt(px, py)
29         {
30             r=pr;
31         }
32         void wypisz() //funkcja przesłania funkcję wypisz()
33                     //z klasy bazowej
34         {
35             punkt::wypisz(); //siegamy po przesłoniętą funkcję
36                             //klasy bazowej, gdyż inaczej nie dostaniemy się
37                             //do jej pól prywatnych
38             cout<<"r = "<<r<<endl;
39         }
40     };
```



Dziedziczenie kilkupokoleniowe

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

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 zajrzeć 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