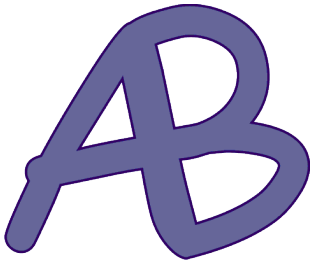
Wykład: Dziedziczenie



Dziedziczenie

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
{ };
```

Dziedziczenie

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;

Dziedziczenie

Klasa nie dziedziczy:

- ✓ Konstruktorów,
- ✓ Destruktora,
- ✓ Operatora przypisania

Dziedziczenie

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.

Dziedziczenie

Typy dziedziczenia:

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

* - niedostępne, jeśli nie ma przyjaźni
```

Dziedziczenie a zasięg widoczności nazw

Przesłanianie dziedziczonych metod

Dziedziczenie

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 **::**

```
Klasa_bazowa::pole;
```

Przesłaniać możemy zarówno pola, jak i metody.

Dziedziczenie

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

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

Dziedziczenie

Inicjalizacja odziedziczonych składowych

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

```
class bazowa
{
private:
  int x;
public:
  bazowa(int parametr) : x(parametr) {} //konstruktor z parametrem
};

class pochodna : public bazowa
{
public:
  pochodna(int parametr) : bazowa(parametr) {}
  // konstruktor klasy pochodnej
  // uruchamia konstruktor klasy bazowej (przekazując do niego parametr),
  // a ten przekazuje parametr dalej, do prywatnego pola klasy bazowej
};
```

Dziedziczenie

Konstruktory klasy pochodnej

- jeżeli istnieje więcej niż jeden konstruktor

```
class bazowa
{
public:
  bazowa() {}                 //konstruktor domyślny
  bazowa(int parametr) {} //konstruktor z parametrem
};

class pochodna : public bazowa
{
public:
  pochodna() : bazowa() {}  //konstruktor klasy pochodnej
                            // uruchamia konstruktor klasy bazowej

  pochodna (int parametr) :bazowa(parametr) {}
};
```

Dziedziczenie

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 funkję wypisz()
33                    //z klasy bazowej
34      {
35          punkt::wypisz(); //siegamy po przesłoniętą funkcje
36                           //klasy bazowej, gdyż inaczej nie dostaniemy się
37                           //do jej pól prywatnych
38          cout<<"r = "<<r<<endl;
39      }
40  };
```

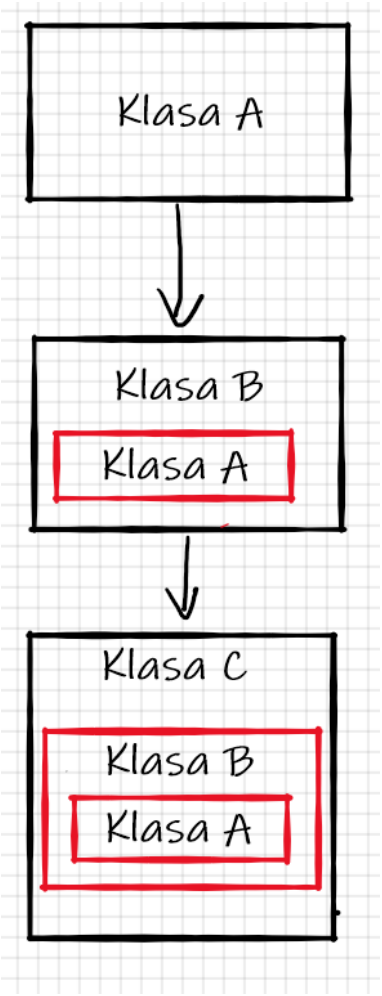
Dziedziczenie

Dziedziczenie wielopokoleniowe

```
class A
{
  // – ciało klasy A
};

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

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



Dziedziczenie

Dziedziczenie wielopokoleniowe - PRZYKŁAD

Klasa „punkt” jest bazą do dziedziczenia. Przechowuje współrzędne punktu na płaszczyźnie i posiada podstawowe metody.

```
#include <iostream>
#include <sstream>

using namespace std;

class punkt
{
protected:
    int x;
    int y;

public:
    punkt(int X, int Y) : x(X), y(Y)
    {
        cout << endl
             << "LOG: konstruktor klasy punkt, z parametrami " << endl;
    }
    void setX(int x) { this->x = x; }
    void setY(int y) { this->y = y; }
    int getX() { return x; }
    int getY() { return y; }
    string toString()
    {
        stringstream temp;
        temp << "(" << x << ";" << y << ")";
        return temp.str();
    }
};
```

Dziedziczenie

```
class punkt_3D : public punkt
{
protected:
    int z;

public:
    punkt_3D(int X, int Y, int Z) : punkt(X, Y), z(Z)
    {
        cout << endl
             << "LOG: konstruktor klasy punkt_3D, z parametrami " << endl;
    }
    void setZ(int z) { this->z = z; }
    int getZ() { return z; }
    string toString()
    {
        stringstream temp;
        temp << "(" << x << ";" << y << ";" << z << ")";
        return temp.str();
    }
};
```

Klasa „punkt_3D” dziedziczy publicznie po klasie „punkt”.

Posiada wszystkie składowe klasy „punkt”.

Dodatkowo uzupełniona jest o:

- trzecią współrzędną „z” (punkt w przestrzeni 3D),
- metody pozwalające na odczyt/modyfikację dodanej współrzędnej,
- nową wersję metody „toString()”

Dziedziczenie

```
class pixel_3D : public punkt_3D
{
protected:
    string kolor;

public:
    pixel_3D(int X, int Y, int Z, string K) : punkt_3D(X, Y, X), kolor(K)
    {
        cout << endl
             << "LOG: konstruktor klasy pixel_3D, z parametrami " << endl;
    }
    void setKolor(string K) { kolor = K; }
    string getKolor() { return kolor; }
    string toString()
    {
        stringstream temp;
        temp << punkt_3D::toString() << "-" << kolor << " ";
        return temp.str();
    }
};

int main()
{
    pixel_3D p1(10, 20, 30, "red");
    cout << p1.toString();
    return 0;
}
```

Klasa „pixel_3D” dziedziczy publicznie po klasie „punkt_3D”, a pośrednio także po klasie „punkt”

Posiada wszystkie składowe klas „punkt” i „punkt_3D”

Dodatkowo uzupełniona jest o:

- Kolor (dla czytelności w postaci nazwy koloru),
- metody pozwalające na odczyt/modyfikację koloru
- nową wersję metody „toString()” Zauważmy, że wywołuje ona jawnie metodę klasy rodzica `punkt_3D::toString()`

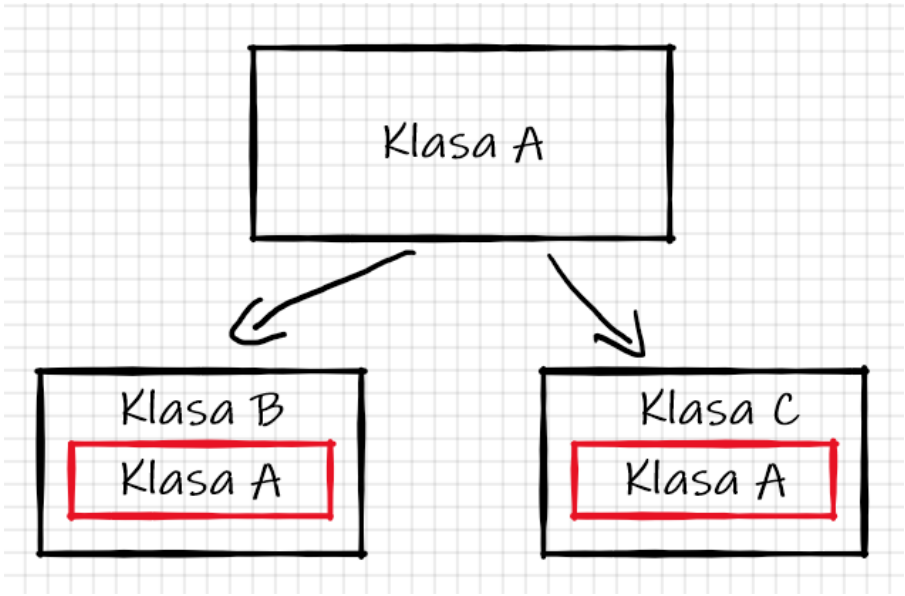
Dziedziczenie

Dziedziczenie równorzędne

```
class A
{
  // – ciało klasy A
};

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

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



Literatura:

W prezentacji wykorzystano przykłady i fragmenty:

- Gręboszj 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.
- Kerninghan B. W., Ritchie D. M.: ***język ANSI C***, Wydawnictwo Naukowo Techniczne.

Dla bardziej zaawansowanych:

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