Programowanie obiektowe

Mechanizm dziedziczenia i hermetyzacji

Destruktor

- Destruktor – specjalna funkcja, która wykonuje czynności dla obiektu, który ma zostać usunięty.
- W danej klasie może być zdefiniowany **tylko jeden** destruktor.

```
24  int main()
25  {
26      RejestratorCzasu r;
27
28      int i;
29      cout << "Wpisz liczbe: ";
30      cin >> i;
31
32      return 0;
33  }
```

Klasa RejestratorCzasu

```cpp
#include <iostream>
#include <ctime>

using namespace std;

class RejestratorCzasu
{
    public:
        RejestratorCzasu()
        {
            start = clock();
        }
        ~RejestratorCzasu()
        {
            clock_t end = clock();
            cout << "Czas: ";
            cout << int( ( end - start )/CLK_TCK );
            cout << " sek.";
        }
    private:
        clock_t start;
};
```

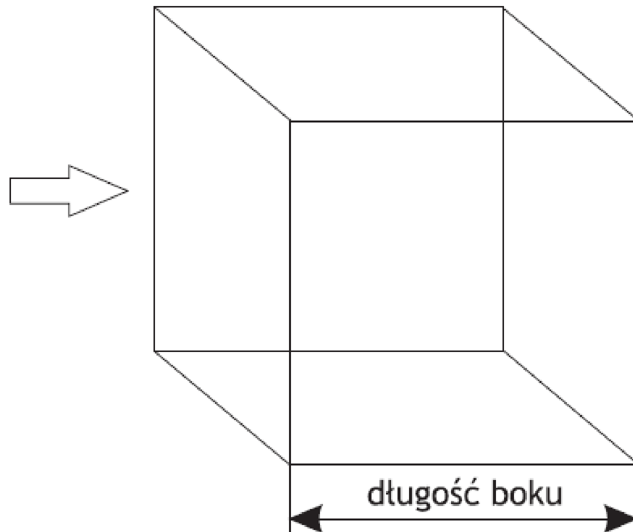
Wstęp do dziedziczenia

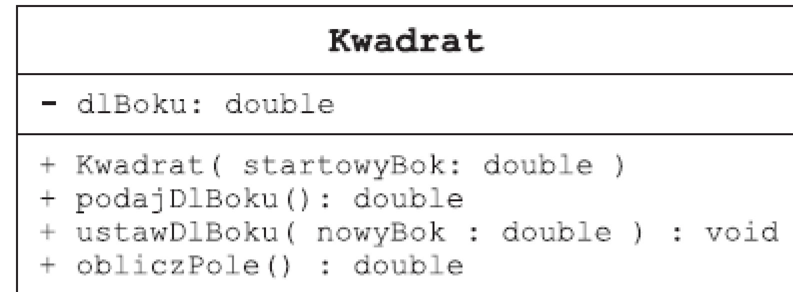
- Wyróżnia się dwa sposoby ponownego wykorzystania klas:
  - Kompozycję – oznacza, że obiekt jest zawarty w innym obiekcie i jest reprezentowana przez relację całość-część. Przykładowo lampa zawiera żarówkę (i nie bardzo sobie można wyobrazić lampę bez żarówki)
  - Dziedziczenie – oznacza przejmowanie własności innego obiektu i poddanie ich wyspecjalizowaniu. Jest reprezentowane przez relację generalizacji-specjalizacji np. Paprotka jest specjalizacją Rośliny.

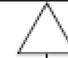
- Technika dziedziczenia umożliwia tworzenie nowych klas na bazie klas już istniejących. Klasa, która dziedziczy po innej klasie, przejmuje jej metody i pola, oraz najczęściej dodaje własne by dostosować ją do konkretnego zadania.
  Wyjątkiem są konstruktory i destruktory – nie są dziedziczone.

- Klasę po której się dziedziczy określa się **nadklasą**, klasą **bazową** lub klasą **macierzystą**.
- Nowo utworzona klasa nazywa się **podklasą**, klasą **pochodną** lub klasą **potomną**.

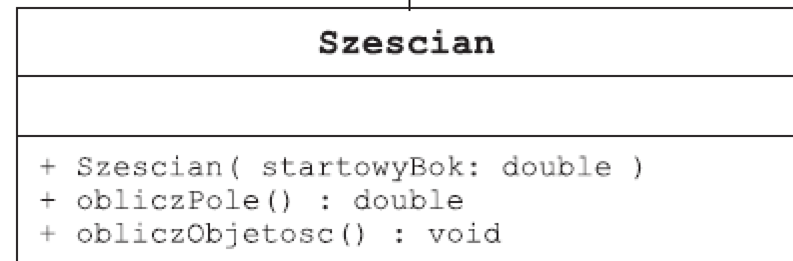
Przykład dziedziczenia – klasa sześcian







Dziedziczenie



Objętość:

$$V = a^3$$

Pole całkowite:

$$P_c = 6a^2$$

Przykład dziedziczenia – klasa sześcian

- **Szescian.h**

```
6   class Szescian : public Kwadrat
7   {
8       public:
9           Szescian( double startowyBok = 0 );
10          double obliczPole();
11          double obliczObjetosc();
12      protected:
13      private:
14  };
```

- **Szescian.cpp**

Tutaj koniecznie lista inicjalizacyjna

```
1   #include "Szescian.h"
2
3   Szescian::Szescian( double startowyBok ) : Kwadrat( startowyBok )
4   {
5   }
6
7   double Szescian::obliczPole()
8   {
9       //return 6 * podajDlBoku() * podajDlBoku();    //Niekoniecznie najlepiej, ale działa
10      return 6 * Kwadrat::obliczPole();
11  }
12
13  double Szescian::obliczObjetosc()
14  {
15      //return podajDlBoku() * podajDlBoku() * podajDlBoku();  //Niekoniecznie najlepiej, ale działa
16      return Kwadrat::obliczPole() * podajDlBoku();
17  }
```

Test klasy Szescian

```
1  #include <iostream>
2  #include "Szescian.h"
3
4  using namespace std;
5
6  int main()
7  {
8      Szescian kostka( 10 );
9      cout << "Szescian o boku: " << kostka.podajDlBoku() << endl;
10     cout << "Objetosc: " << kostka.obliczObjetosc() << endl;
11     cout << "Powierzchnia: " << kostka.obliczPole() << endl;
12     return 0;
13 }
14
```

Redefinicja metod –niebezpieczeństwa

- Załóżmy, że programista pomylił się przy redefinicji metody obliczPole():

```
double Szescian::obliczPole()
{
  return 6 * obliczPole();
}
```

- **Błąd!** Nie jest określone, że ma zostać wywołana funkcja obliczPole() z klasy Kwadrat. Co więc się stanie?
- Zostanie wywołana rekurencyjnie funkcja obliczPole() z klasy Szescian.

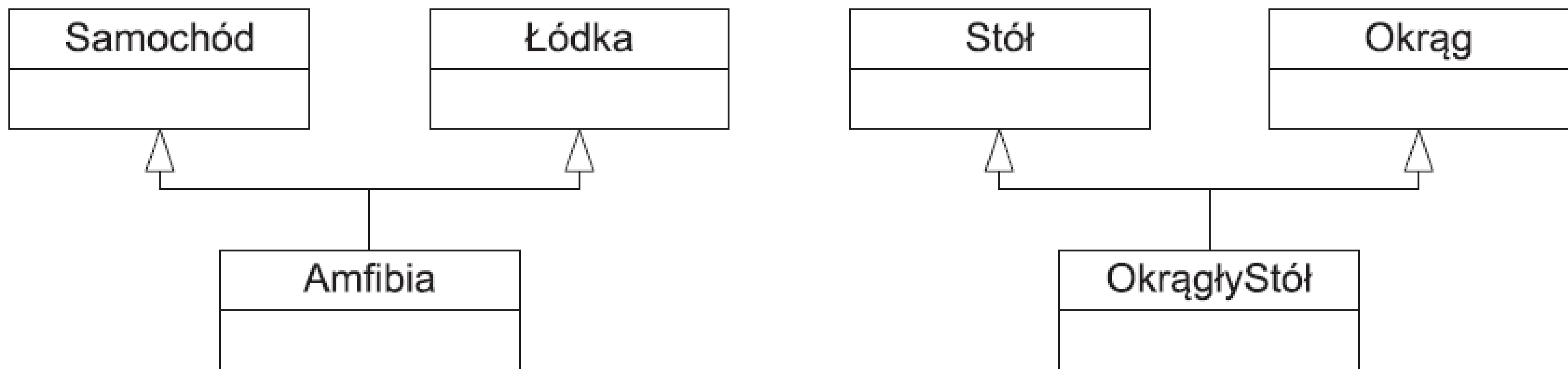
Zakres widoczności protected

- Zasady rządzące sekcją `protected`:
  - Składowe zadeklarowane jako **protected** są dostępne dla obiektów wszystkich klas pochodnych (tak jak składowe **public**).
  - Składowe zadeklarowane jako **protected** są niedostępne dla obiektów innych, niezaprzyjaźnionych klas (tak jak składowe **private**).
  - Specyfikator **protected** działa jak **private**, z tym wyjątkiem, że obiekty klas pochodnych otrzymują dostęp do **składowych protected klasy bazowej**.
  - Pola i funkcje zadeklarowane w sekcji **protected** nazywane są *chronionymi.*

```
class Kwadrat
{
   . . .
   protected:
        double dlBoku;
};

double Szescian::obliczObjetosc()
{
     return Kwadrat::obliczPole() * dlBoku;
}
```

Dziedziczenie wielobazowe




```
 3  const double PI = 3.1416;
 4
 5  class Okrag
 6  {
 7      public:
 8          Okrag( double r = 0 ) : promien( r ) {}
 9          double obliczPole()
10          {
11              return PI * promien * promien;
12          }
13      protected:
14          double promien;
15  };
```

Dziedziczenie wielobazowe c.d.

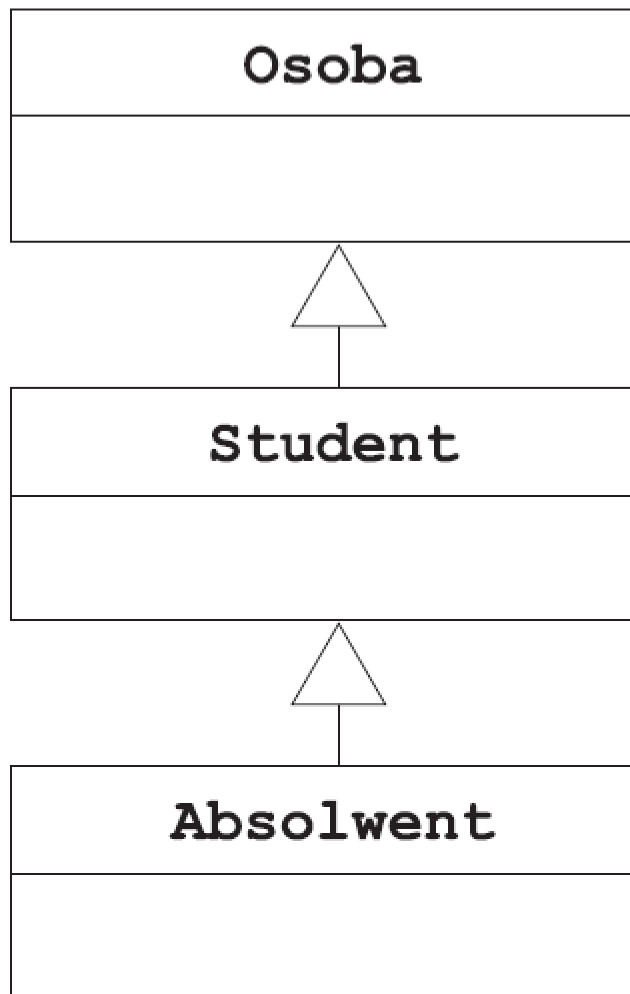
```
17  class Stol
18  {
19      public:
20          Stol( int ln = 0 ) : liczbaNog( ln ) {}
21          int podajLiczbeNog()
22          {
23              return liczbaNog;
24          }
25      protected:
26          int liczbaNog;
27  };
28
29  class OkraglyStol : public Okrag, public Stol
30  {
31  public:
32      OkraglyStol( double r = 0, int ln = 0 )
33      : Okrag( r ), Stol( ln ) { }
34  };
```

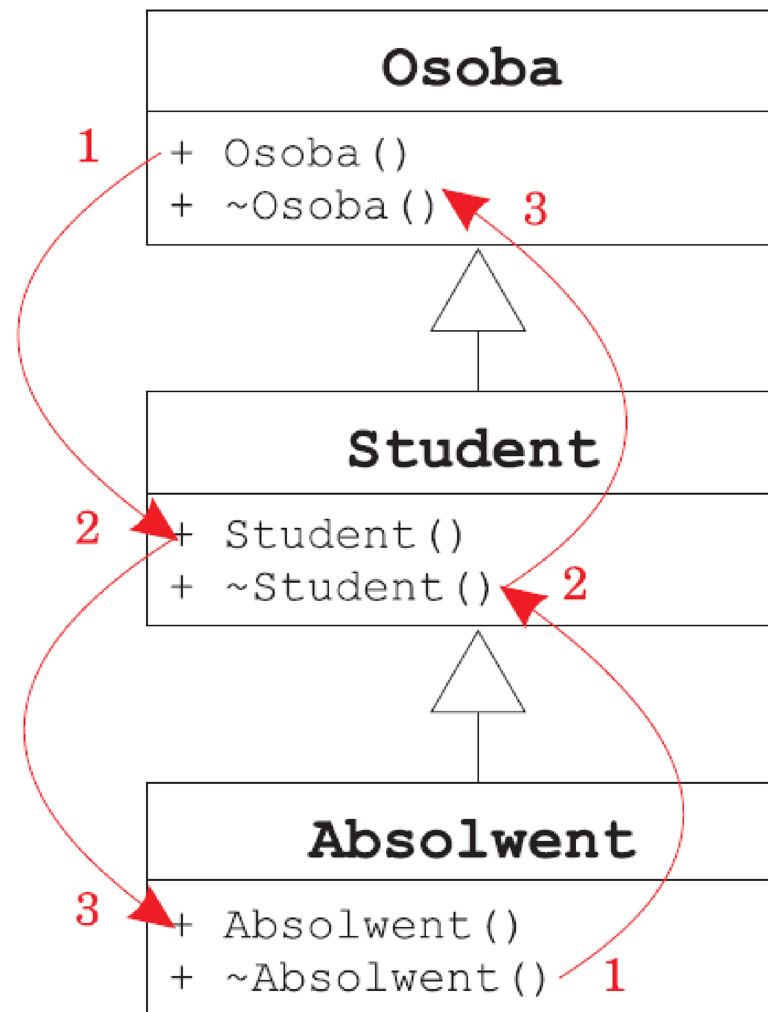
Dziedziczenie wielobazowe c.d.

```
36   int main()
37   {
38       OkraglyStol stolik( 1, 3 ); // 1 - promien, 3 - lb. nog
39       cout << "\nLiczba nog : " << stolik.podajLiczbeNog();
40       cout << "\nPowierzchnia stolu : " << stolik.obliczPole();
41       return 0;
42   }
```

- Dla obiektu stolik aktywowany jest konstruktor klasy **OkraglyStol**.
- Ponieważ klasa ta dziedziczy po dwóch klasach (**Stol**, **Okrag**), na liście inicjalizacyjnej tego konstruktora umieszczone są odwołania do dwóch konstruktorów.
- W trakcie przetwarzania listy inicjalizacyjnej następuje aktywowanie obu tych konstruktorów.
- Kolejność aktywowania konstruktorów dla obiektu klasy pochodnej wynika z **kolejności występowania nazw klas bazowych w deklaracji** tej klasy. Nie jest istotna kolejność ich umieszczenia na liście inicjalizacyjnej konstruktora klasy pochodnej.

Konstruktory, destruktory, a hierarchia klas.





Hierarchia klas – przykład

- Proszę pobrać projekt pod adresem:

http://www.tomaszx.pl/materialy/JpoHierarchiaKlas.zip
oraz zapoznać się z jego zawartością.

Następnie proszę odpowiedzieć na następujące pytania:

- Jaka jest kolejność aktywacji konstruktorów i destruktorów, gdy obiekt tworzony jest jako `Absolwent a;` ?
- Jaka jest kolejność aktywacji konstruktorów i destruktorów, gdy obiekt tworzony jest jako `Absolwent a( 10 );` ?

Proszę **usunąć aktywację konstruktorów klas bazowych** z listy inicjalizacyjnej konstruktorów klas pochodnych Absolwent i Student oraz sprawdzić jak sytuacji wygląda po rekompilacji programu.

Ćwiczenia 1, 2

1. Bazując na klasie Data (stworzonej wcześniej) zdefiniować z wykorzystaniem dziedziczenia klasę **DataZKontrola** — w tej klasie należy dokonać redefinicji funkcji ustawiających wartości pól, tak by kontrolowały one poprawność wartości przekazywanych parametrów (roku, miesiąca, dnia).

2. Bazując na klasie Punkt (stworzonej wcześniej) oraz wykorzystując mechanizm dziedziczenia, zdefiniować klasę **Punkt3D**, reprezentującą punkt w przestrzeni trójwymiarowej. Klasa ta powinna posiadać dodatkowe pole **z** przechowujące współrzędną trzeciego wymiaru, odpowiednie funkcje dostępowe **ustawZ**, **pobierzZ**, własny konstruktor bezparametrowy i parametrowy.

Ćwiczenia 3

3. Bazując na klasach opisu figur płaskich stworzonych wcześniej, zdefiniować ich klasy pochodne, reprezentujące bryły: **sześcian**, **prostopadłościan**, **kula**, **graniastosłup** o podstawie **trapezu.** W klasach pochodnych należy dodać wszelkie informacje konieczne dla **obliczenia pól** tych brył oraz należy **przedefiniować** funkcje składowe obliczania pola (funkcje **obliczPole** każdej z klas reprezentujących figurę płaską), tak by wyznaczały właściwe pola brył.

Ćwiczenia 4

4. Poniżej podana została klasa Lista, która służy do przechowywania listy liczb całkowitych, uzupełnij metody tej klasy.

```
class Lista {
 private:
     int liczby[100]; // tablica na przechowywane liczby
     int pojemnosc;   // liczba elementów, które można
                      //  pomieścić na liście
     int rozmiar;     //  liczba elementów aktualnie
                      //  przechowywana na liście
 public:
     Lista();

  // dodaje nową liczbę do listy
     void dodaj(int liczba);
  // zwraca element listy znajdujący sie na podanym indeksie
     int dajElement(int indeks);
  // wyświetla aktualny rozmiar listy, tj. liczbę elementów
     int wezRozmiar();
 };
```

Ćwiczenia 4

4. c.d. Przykład użycia klasy lista:

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "Lista.h"

using namespace std;

int main()
{
    Lista l;
    srand(time(NULL));
    for(int i = 0; i < 20; i++)
        l.dodaj( rand() % 100 );

    cout << "Zawartosc listy: ";
    for(int i = 0; i < l.wezRozmiar(); i++)
        cout << l.dajElement(i) << " ";
    cout << endl;

    return 0;
}
```

Ćwiczenia 5

5. Zdefiniuj klasę **Przedmiot** reprezentującą przedmiot ćwiczeniowy studenta. Klasa powinna zawierać:
   - pola prywatne:
     - **double * oceny;** *// oceny studenta*
     - **int nr_indeksu;** *// numer indeksu studenta*
     - **int liczba_ocen;** *// liczba wprowadzonych ocen*
     - **int limit_ocen;** *// maksymalna liczba ocen*
   - publiczny konstruktor z dwoma parametrami:
     - **int nr_indeksu, int limit_ocen**
   - metody publiczne:
     - **double obliczSrednia();** *// oblicza średnią arytmetyczną ocen*
     - **bool zaliczenie();** *// określa, czy student otrzyma zaliczenie*
     - **void dodajOcene(double ocena);** *// dodaje nową ocenę*
     - **void zmienOcene(double ocena, int id);** *// zmienia ocenę*
     - **void pisz();** *// wypisuje na ekranie informację*

Ćwiczenia 5

5. Dodatkowe informacje:

- Funkcja **obliczSrednia** oblicza i zwraca średnią arytmetyczną wpisanych już ocen
- Funkcja **zaliczenie** ustala, czy student otrzyma zaliczenie (średnia wszystkich uzupełnionych ocen większa lub równa 3.0). Jeśli otrzyma, to zwraca **true**, w przeciwnym razie **false**
- Funkcja **dodajOcene** dodaje nową ocenę (jeśli jest to możliwe – nie przekroczy limitu)
- Funkcja **zmienOcene z**mienia ocenę na podanej pozycji (id liczonej od 0), ale tylko jeśli taka ocena istnieje;
- Funkcja **pisz** wyświetla na ekran studenta, wszystkie uzupełnione oceny i informację, czy aktualnie dostałby zaliczenie (tak lub nie).

Ćwiczenia 6

6. Załóżmy, że chcemy stworzyć program zarządzający pracownikami firmy X.

- W programie należy zapamiętać nazwisko pracownika i numer pokoju w którym przebywa. Na podstawie numeru pokoju odpowiednia funkcja ma wyświetlić numer telefonu (np. pokój 201 telefon 8431**201**).

Firma X zatrudnia różnych pracowników (każdy to osobna klasa):

- **Pracownicy** pracujący na akord. Pensja pracownika równa się liczbie przepracowanej godzin razy stawka godzinowa.
- **Sprzedawcy**. Wypłata równa się stałej pensji (wyliczanej jak dla pracowników na akord) + dodatek wyliczony jako procent zrealizowanej w danym miesiącu sprzedaży.
- **Dyrektorzy**. Wypłata równa się stałej pensji (ustalonej odgórnie) oraz premii uzależnionej od liczby podległych mu pracowników. Każdy pracownik to 100 zł więcej premii.

Każda klasa (Pracownik, Sprzedawca, Dyrektor) musi dziedziczyć z ogólnej klasy Personel (zawierającej nazwisko, numer pokoju i funkcję wyznaczającą numer telefonu).

Należy pamiętać o zachowaniu pełnej hermetyzacji.