Interfejs klasy, konstruktory i destruktory, lista inicjalizacyjna, dziedziczenie

Bogdan Kreczmer

bogdan.kreczmer@pwr.edu.pl

Katedra Cybernetyki i Robotyki
Wydziału Elektroniki
Politechnika Wrocławska

Kurs: Programowanie obiektowe

Niniejszy dokument zawiera materiały do wykładu dotyczącego programowania obiektowego. Jest on udostępniony pod warunkiem wykorzystania wyłącznie do własnych prywatnych potrzeb i może on być kopiowany

Niniejsza prezentacja została wykonana przy użyciu systemu składu LaTeX *oraz stylu* `beamer`*, którego autorem jest Till Tantau.*

Strona domowa projektu `Beamer`:
`http://latex-beamer.sourceforge.net`

Plan prezentacji

Plan prezentacji

Potencjalne możliwości błędów

```
struct Stos { //.....
    float _TabStosu[ROZ_STOSU];
    int   _Ilosc;

    Stos( ) { _Ilosc = 0; }
}; //.....
```

Potencjalne możliwości błędów

```
struct Stos { //.....
    float _TabStosu[ROZ_STOSU];
    int   _Ilosc;

    Stos( ) { _Ilosc = 0; }
}; //.....

int main( )
{
    Stos St;
    float Arg;
    ...
    if (St._Ilosc >= ROZ_STOSU) { /* Obsługa błędu */ }
    St._TabStosu[St._Ilosc++] = 12.3;
    ...
}
```

Potencjalne możliwości błędów

```
struct Stos { //.....
    float _TabStosu[ROZ_STOSU];
    int   _Ilosc;

    Stos( ) { _Ilosc = 0; }
}; //.....

int main( )
{
    Stos St;
    float Arg;
    ...
    if (St._Ilosc >= ROZ_STOSU) { /* Obsługa błędu */ }
    St._TabStosu[St._Ilosc] = 12.3;
    ...
}
```

Potencjalne możliwości błędów

```
struct Stos { //.....
    float _TabStosu[ROZ_STOSU];
    int   _Ilosc;

    Stos( ) { _Ilosc = 0; }
}; //.....

int main( )
{
    Stos St;
    float Arg;
    ...
    if (St._Ilosc <= 0) { /* Obsługa błędu */ }
    Arg = St._TabStosu[--St._Ilosc];
    ...
}
```

Potencjalne możliwości błędów

```
struct Stos { //.....
    float _TabStosu[ROZ_STOSU];
    int   _Ilosc;

    Stos( ) { _Ilosc = 0; }
}; //.....

int main( )
{
    Stos St;
    float Arg;
    ...
    if (St._Ilosc <= 0) { /* Obsługa błędu */ }
    Arg = St._TabStosu[-St._Ilosc];
    ...
}
```

Plan prezentacji

Metoda jako element interfejsu klasy

```
struct Stos { //.....
    float _TabStosu[ROZ_STOSU];
    int   _Ilosc;

    Stos( ) { _Ilosc = 0; }
}; //.....
```

Metoda jako element interfejsu klasy

```
struct Stos { //.....
    float _TabStosu[ROZ_STOSU];
    int   _Ilosc;

    Stos( ) { _Ilosc = 0; }
    float Pobierz( );
}; //.....
```

Metoda jako element interfejsu klasy

```
struct Stos { //.....
    float _TabStosu[ROZ_STOSU];
    int   _Ilosc;

    Stos( ) { _Ilosc = 0; }
    float Pobierz( );
}; //.....

float Stos::Pobierz( )
{
  return  _TabStosu[--_Ilosc];
}
```

Metoda jako element interfejsu klasy

```
struct Stos { //.....
    float _TabStosu[ROZ_STOSU];
    int   _Ilosc;

    Stos( ) { _Ilosc = 0; }
    float Pobierz( );
}; //.....

float Stos::Pobierz( )
{
  if  (_Ilosc <= 0) return  numeric_limits<float>::max( );
  return  _TabStosu[--_Ilosc];
}
```

Metoda jako element interfejsu klasy

```
#include <limits>
using namespace std;

struct Stos { // ..........................
    float _TabStosu[ROZ_STOSU];
    int   _Ilosc;

    Stos( ) { _Ilosc = 0; }
    float Pobierz( );
}; //.....................................................................

float Stos::Pobierz( )
{
  if (_Ilosc <= 0) return numeric_limits<float>::max( );
  return _TabStosu[--_Ilosc];
}
```

Aby móc korzystać z danych zawierających informacje o typach wbudowanych, należy dołączyć plik nagłówkowy `limits`.

Metoda jako element interfejsu klasy

```
struct Stos { //.....
    float  _TabStosu[ROZ_STOSU];
    int    _Ilosc;
    ...
}; //.....

float Stos::Pobierz( )
{
  if  (_Ilosc <= 0) return  numeric_limits<float>::max( );
  return  _TabStosu[--_Ilosc];
}

int main( )
{
    ...
    Arg = St.Pobierz( );
    if  (Arg == numeric_limits <float>::max()) { /* Obsługa błędu */ }
    ...
}
```

Przeanalizujmy sposób wykorzystania metody `Pobierz` pod względem jej konstrukcji. Chcemy, aby zapewniła ona jak najlepszy zapis użycia operacji pobrania elementu ze stosu.

Metoda jako element interfejsu klasy

```
struct Stos { //.....
      float  _TabStosu[ROZ_STOSU];
      int    _Ilosc;
      ...
}; //.....

float  Stos::Pobierz( )
{
   if   (_Ilosc <= 0) return  numeric_limits<float>::max( );
   return   _TabStosu[--_Ilosc];
}

 int  main( )
{
      ...
      Arg = St.Pobierz( );
      if   (Arg == numeric_limits <float>::max()) { /* Obsługa błędu */ }
      ...
}
```

Metoda jako element interfejsu klasy

```
struct Stos { //.....
    float _TabStosu[ROZ_STOSU];
    int   _Ilosc;
    ...
}; //.....

float Stos::Pobierz(bool & Sukces )
{
  if (_Ilosc <= 0) { Sukces = false; return numeric_limits<float>::max( ); }
  return Sukces = true, _TabStosu[--_Ilosc];
}

int main( )
{
    ...
    Arg = St.Pobierz(CzySukces);
    if (CzySukces) { /* Obsługa błędu */ }
    ...
}
```

Metoda jako element interfejsu klasy

```
struct Stos { //.....
    float  _TabStosu[ROZ_STOSU];
    int    _Ilosc;
    ...
}; //.....

float  Stos::Pobierz(bool & Sukces )
{
  if  (_Ilosc <= 0) { Sukces = false; return  numeric_limits<float>::max( ); }
  return  Sukces = true, _TabStosu[--_Ilosc];
}

 int  main( )
{
    ...
    Arg = St.Pobierz(CzySukces);
    if  (CzySukces) { /* Obsługa błędu */ }
    ...
}
```

Metoda jako element interfejsu klasy

```
struct Stos { //.....
    float _TabStosu[ROZ_STOSU];
    int   _Ilosc;
    ...
}; //.....

float Stos::Pobierz(bool & Sukces )
{
  if (_Ilosc <= 0) { Sukces = false; return numeric_limits<float>::max( ); }
  return Sukces = true, _TabStosu[--_Ilosc];
}

int main( )
{
    ...
    Arg = St.Pobierz(CzySukces);
    if (CzySukces) { /* Obsługa błędu */ }
    ...
}
```

Metoda jako element interfejsu klasy

```
struct Stos { //.....
      float  _TabStosu[ROZ_STOSU];
       int     _Ilosc;
      ...
}; //.....

bool  Stos::Pobierz(float & ZeStosu )
{
   if   (_Ilosc <= 0) return  false;
   return  ZeStosu = _TabStosu[--_Ilosc], true;
}

 int  main( )
{
      ...
      Arg = St.Pobierz(CzySukces);
      if   (CzySukces) { /* Obsługa błędu */ }
      ...
}
```

Metoda jako element interfejsu klasy

```
struct Stos { //.....
      float  _TabStosu[ROZ_STOSU];
       int     _Ilosc;
      ...
}; //.....

bool  Stos::Pobierz(float & ZeStosu )
{
   if   (_Ilosc <= 0) return  false;
   return  ZeStosu = _TabStosu[--_Ilosc], true;
}

 int  main( )
{
      ...
      if   (St.Pobierz(Arg)) { /* Obsługa błędu */ }
      ...
}
```

Metoda jako element interfejsu klasy

```
struct Stos { //.....
      float  _TabStosu[ROZ_STOSU];
       int     _Ilosc;
      ...
}; //.....

bool  Stos::Pobierz(float & ZeStosu )
{
   if   (_Ilosc <= 0) return  false;
   return  ZeStosu = _TabStosu[--_Ilosc], true;
}

 int  main( )
{
      ...
      if   (St.Pobierz(Arg)) { /* Obsługa błędu */ }
      ...
}
```

Czy takie rozwiązanie pozwala w pełni zabezpieczyć się przed błędnym pobraniem wartości ze stosu?

Metoda jako element interfejsu klasy

```
struct Stos { //.....
      float  _TabStosu[ROZ_STOSU];
       int     _Ilosc;
      ...
}; //.....

bool  Stos::Pobierz(float & ZeStosu )
{
   if   (_Ilosc <= 0 || ROZ_STOSU < _Ilosc) return  false;
   return  ZeStosu = _TabStosu[--_Ilosc], true;
}

 int  main( )
{
      ...
      if   (St.Pobierz(Arg)) { /* Obsługa błędu */ }
      ...
}
```

Metoda jako element interfejsu klasy

```
struct Stos { //.....
    float  _TabStosu[ROZ_STOSU];
    unsigned int    _Ilosc;
    ...
}; //.....

bool  Stos::Pobierz(float & ZeStosu )
{
   if   (_Ilosc <= 0 || ROZ_STOSU < _Ilosc) return  false;
   return  ZeStosu = _TabStosu[--_Ilosc], true;
}

 int  main( )
{
    ...
    if   (St.Pobierz(Arg)) { /* Obsługa błędu */ }
    ...
}
```

Metoda jako element interfejsu klasy

```
struct Stos { //.....
    float  _TabStosu[ROZ_STOSU];
    unsigned int    _Ilosc;
    ...
}; //.....

bool  Stos::Pobierz(float & ZeStosu )
{
   if  (!_Ilosc  || ROZ_STOSU < _Ilosc) return  false;
   return  ZeStosu = _TabStosu[--_Ilosc], true;
}

 int  main( )
{
    ...
    if  (St.Pobierz(Arg)) { /* Obsługa błędu */ }
    ...
}
```

Metoda jako element interfejsu klasy

```
struct Stos { //.....
    float _TabStosu[ROZ_STOSU];
    unsigned int  _Ilosc;
    bool Pobierz(float& ZeStosu);
    bool Poloz(float NaStos);
}; //.....

  ...

int main( )
{
    ...
    if (St.Pobierz(Arg)) { /* Obsługa błędu */ }
    ...
}
```

Metoda jako element interfejsu klasy

```
struct Stos { //.....
    float  _TabStosu[ROZ_STOSU];
    unsigned int     _Ilosc;
    bool  Pobierz(float&  ZeStosu);
    bool  Poloz(float NaStos);
}; //.....

  ...

int  main( )
{
    ...
    if   (St.Pobierz(Arg)) { /* Obsługa błędu */ }
    St._TabStosu[3] = 5;
    --St._Ilosc;
    ...
}
```

Plan prezentacji

Metoda jako element interfejsu klasy

```
struct Stos { //.....
    bool  Pobierz(float&  ZeStosu);
    bool  Poloz(float NaStos);
  private:
    float  _TabStosu[ROZ_STOSU];
    unsigned int     _Ilosc;
}; //.....

  ...

int  main( )
{
    ...
    if  (St.Pobierz(Arg)) { /* Obsługa błędu */ }
    St._TabStosu[3] = 5;
    --St._Ilosc;
    ...
}
```

Metoda jako element interfejsu klasy

```
struct Stos { //.....
    bool Pobierz(float& ZeStosu);
    bool Poloz(float NaStos);
  private:
    float _TabStosu[ROZ_STOSU];
    unsigned int  _Ilosc;
}; //.....

  ...

int main( )
{
    ...
    if (St.Pobierz(Arg)) { /* Obsługa błędu */ }
    St._TabStosu[3] = 5;
    --St._Ilosc;
    ...
}
```

Metoda jako element interfejsu klasy

```
struct Stos { //.....
    bool Pobierz(float& ZeStosu);
    bool Poloz(float NaStos);
  private:
    float _TabStosu[ROZ_STOSU];
    unsigned int    _Ilosc;
}; //.....

  ...


 int main( )
{
    ...
    if (St.Pobierz(Arg)) { /* Obsługa błędu */ }
    ...
}
```

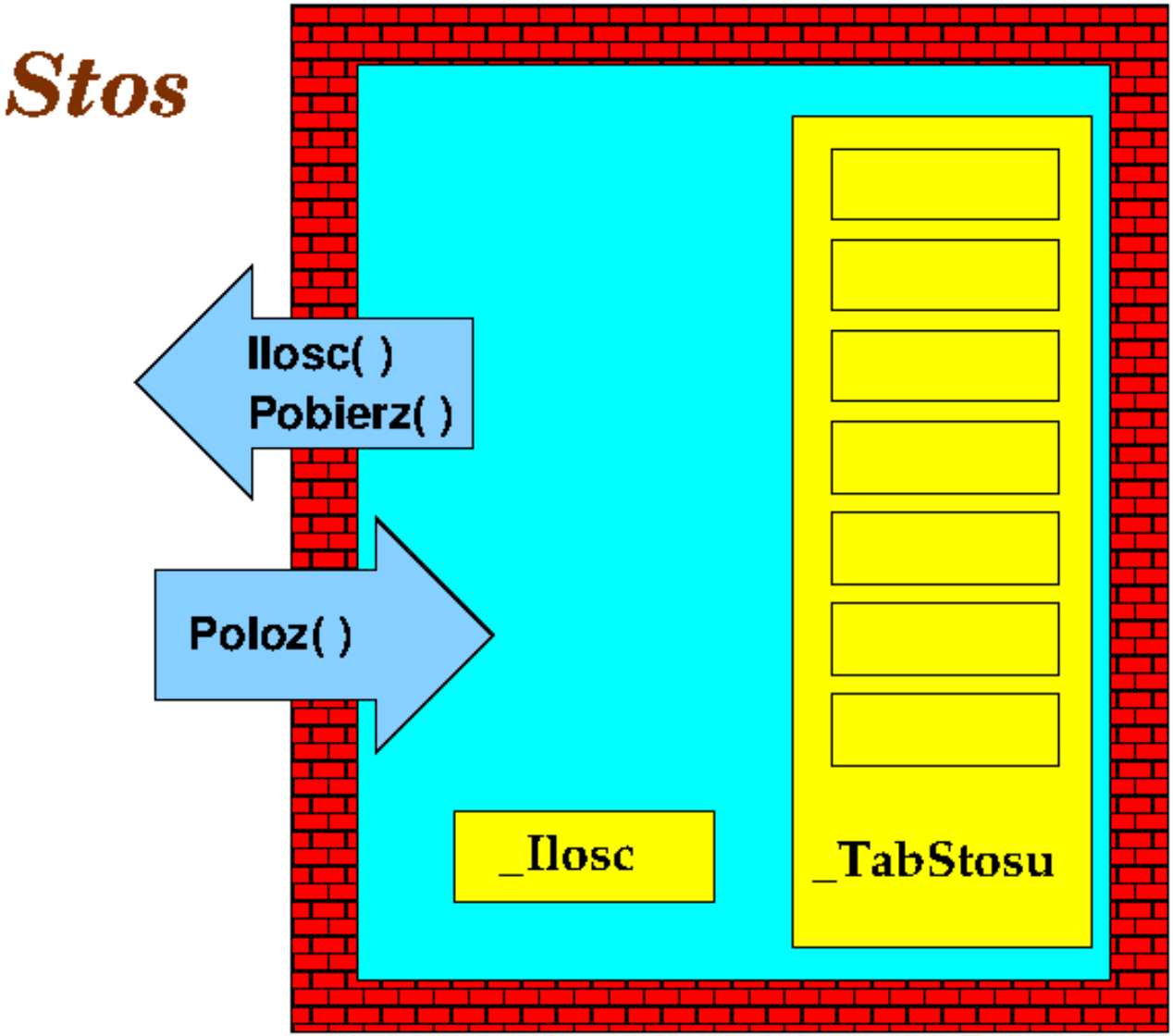
Metoda jako element interfejsu klasy

```
struct Stos { //.....
    bool  Pobierz(float&  ZeStosu);
    bool  Poloz(float NaStos);
    unsigned int Ilosc( ) { return  _Ilosc; }
  private:
    float  _TabStosu[ROZ_STOSU];
    unsigned int     _Ilosc;
}; //.....

   ...


 int  main( )
{
    ...
    if  (St.Pobierz(Arg)) { /* Obsługa błędu */ }
    ...
}
```

Metoda jako element interfejsu klasy

```
struct Stos { //.....
    bool  Pobierz(float&  ZeStosu);
    bool  Poloz(float NaStos);
    unsigned int Ilosc( ) const { return  _Ilosc; }
  private:
    float  _TabStosu[ROZ_STOSU];
    unsigned int     _Ilosc;
}; //.....

  ...


 int  main( )
{
    ...
    if  (St.Pobierz(Arg)) { /* Obsługa błędu */ }
    ...
}
```

Interfejs klasy



Podsumowanie: Ważniejsze cechy i własności

- Hermetyzacja pozwala ukryć wewnętrzne cechy implementacji danej klasy.
- Daje możliwość wydzielenia tej części interfejsu, która realizuje modyfikację obiektu oraz tej, która służy do odczytu stanu obiektu.
- Umożliwia zdefiniowanie interfejsu, który określa sposób manipulowania wewnętrznymi strukturami obiektu. *Pozwala to na zapewnienie spójności wewnętrznych struktur danych.*
- Zapobiega przypadkowej ingerencji w wewnętrzną strukturę obiektu, która mogłaby spowodować utratę spójność przechowywanych danych w obiekcie.

Podsumowanie: Ważniejsze cechy i własności

- Hermetyzacja pozwala ukryć wewnętrzne cechy implementacji danej klasy.
- Daje możliwość wydzielenia tej części interfejsu, która realizuje modyfikację obiektu oraz tej, która służy do odczytu stanu obiektu.
- Umożliwia zdefiniowanie interfejsu, który określa sposób manipulowania wewnętrznymi strukturami obiektu. *Pozwala to na zapewnienie spójności wewnętrznych struktur danych.*
- Zapobiega przypadkowej ingerencji w wewnętrzną strukturę obiektu, która mogłaby spowodować utratę spójność przechowywanych danych w obiekcie.

Podsumowanie: Ważniejsze cechy i własności

- Hermetyzacja pozwala ukryć wewnętrzne cechy implementacji danej klasy.
- Daje możliwość wydzielenia tej części interfejsu, która realizuje modyfikację obiektu oraz tej, która służy do odczytu stanu obiektu.
- Umożliwia zdefiniowanie interfejsu, który określa sposób manipulowania wewnętrznymi strukturami obiektu. *Pozwala to na zapewnienie spójności wewnętrznych struktur danych.*
- Zapobiega przypadkowej ingerencji w wewnętrzną strukturę obiektu, która mogłaby spowodować utratę spójność przechowywanych danych w obiekcie.

Podsumowanie: Ważniejsze cechy i własności

- Hermetyzacja pozwala ukryć wewnętrzne cechy implementacji danej klasy.
- Daje możliwość wydzielenia tej części interfejsu, która realizuje modyfikację obiektu oraz tej, która służy do odczytu stanu obiektu.
- Umożliwia zdefiniowanie interfejsu, który określa sposób manipulowania wewnętrznymi strukturami obiektu. *Pozwala to na zapewnienie spójności wewnętrznych struktur danych.*
- Zapobiega przypadkowej ingerencji w wewnętrzną strukturę obiektu, która mogłaby spowodować utratę spójność przechowywanych danych w obiekcie.

Plan prezentacji

Formułowanie warunków

```
 Licznik = 0;
do   {
        ...
        ++Licznik;
        ...
        ...
        ...
} while   (Licznik != MAKS_ILOSC);
        ...
if   (Licznik == MAKS_ILOSC) cerr <<  "Blad" <<  endl;
```

Formułowanie warunków

```
 Licznik = 0;
do   {
       ...
       ++Licznik;
       ...
       if   ( JakisWarunek ) ++Licznik;
       ...
} while   (Licznik != MAKS_ILOSC);
       ...
if   (Licznik == MAKS_ILOSC) cerr <<  "Blad" <<  endl;
```

Formułowanie warunków

```
 Licznik = 0;
do   {
       ...
       ++Licznik;
       ...
       if   ( JakisWarunek ) ++Licznik;
       ...
} while   (Licznik != MAKS_ILOSC);
       ...
if   (Licznik == MAKS_ILOSC) cerr <<  "Blad" <<  endl;
```

Formułowanie warunków

```
Licznik = 0;
do {
    ...
    ++Licznik;
    ...
    if ( JakisWarunek ) ++Licznik;
    ...
} while (Licznik < MAKS_ILOSC);
    ...
if (Licznik >= MAKS_ILOSC) cerr << "Blad" << endl;
```

Plan prezentacji

Konstruktory i destruktory

```
struct PustaKlasa { //.....

        PustaKlasa( ) { cout << "++PustaKlasa" << endl; }
        ~PustaKlasa( ) { cout << "--PustaKlasa" << endl; }
}; //.....


int  main( )
{
      cout <<  "Witajcie !!! :-)" <<  endl;
      PustaKlasa  Ob;
      cout <<  "Zegnajcie !!! :-(" <<  endl;
}
```

Wynik działania

```
Witajcie !!! :-)
++PustaKlasa
Zegnajcie !!! :-(
--PustaKlasa
```

W przypadku zmiennych lokalnych (automatycznych) konstruktor wywoływany jest w chwili tworzenia obiektu, tj. w miejscu jego definicji. Destruktor natomiast jest wywoływany w miejscu końca zakresu ważności definicji obiektu.

Plan prezentacji

Obiekty tworzone dynamicznie

```
struct PustaKlasa { //.....................................................
    PustaKlasa( ) { cout << "++PustaKlasa" << endl; }
    ~PustaKlasa( ) { cout << "--PustaKlasa" << endl; }
}; //.....................................................................
```

```
int main( )
{
    cout << "Witajcie !!! :-)" << endl;
    PustaKlasa *wOb = new PustaKlasa;
    cout << "Zegnajcie !!! :-(" << endl;
}
```

Wynik działania

```
Witajcie !!! :-)
++PustaKlasa
Zegnajcie !!! :-(
```

W przypadku obiektów tworzonych w sposób dynamiczny, to programista decyduje kiedy je zniszczyć. Zakończenie programu bez wykonania tej operacji nie spowoduje automatycznego uruchomienia destruktorów tych obiektów.

Obiekty tworzone dynamicznie

```
struct PustaKlasa { //.....
    PustaKlasa( ) { cout << "++PustaKlasa" << endl; }
    ~PustaKlasa( ) { cout << "--PustaKlasa" << endl; }
}; //.....
```

```
int main( )
{
    cout << "Witajcie !!! :-)" << endl;
    PustaKlasa *wOb = new PustaKlasa;
    delete wOb;
    cout << "Zegnajcie !!! :-(" << endl;
}
```

Wynik działania

```
Witajcie !!! :-)
++PustaKlasa
--PustaKlasa
Zegnajcie !!! :-(
```

Operator **delete** można zastosować tylko do zmiennej wskaźnikowej w dowolnym miejscu programu. Jedynym warunkiem poprawności jego wywołania jest to, aby pod danym adresem istniał obiekt utworzony dynamicznie z wykorzystaniem operatora **new**, albo też zmienna wskaźnikowa miała wartość **nullptr**.

Tablice obiektów tworzone dynamicznie

```
struct PustaKlasa { //.....................................................

    PustaKlasa( ) { }
    ~PustaKlasa( ) { }
}; //.....................................................................


int  main( )
{
     int          *wTab = new  int[ 30 ];
     PustaKlasa   *wOb =  new  PustaKlasa[ 20 ];

     delete  [ ]  wTab;
     delete  [ ]  wOb;
}
```

W przypadku tworzenia w sposób dynamiczny tablic obiektów stosując operator **delete** należy jawnie wskazać, że usuwana jest tablica obiektów, a nie pojedynczy obiekt. Rozmiar tablicy nie jest istotny.

Plan prezentacji

Konstruktory w klasach złożonych

```
struct KolaPrzednie { //.....
    KolaPrzednie( ) { cout << "++KolaPrzednie" << endl; }
}; //.....

struct KolaTylnie { //.....
    KolaTylnie( ) { cout << "++KolaTylnie" << endl; }
}; //.....

struct Pojazd4Kolowy { //.....
    KolaPrzednie   _KolaPrz;
    KolaTylnie     _KolaTyl;
    Pojazd4Kolowy( ) { cout << "++Pojazd4Kolowy" << endl; }
}; //.....

int main( )
{
    Pojazd4Kolowy   Ob;
}
```

Konstruktory w klasach złożonych

```
struct KolaPrzednie { //.....
    KolaPrzednie( ) { cout << "++KolaPrzednie" << endl; }
}; //.....

struct KolaTylnie { //.....
    KolaTylnie( ) { cout << "++KolaTylnie" << endl; }
}; //.....

struct Pojazd4Kolowy { //.....
    KolaPrzednie  _KolaPrz;
    KolaTylnie    _KolaTyl;
    Pojazd4Kolowy( ) { cout << "++Pojazd4Kolowy" << endl; }
}; //.....

int main( )
{
    Pojazd4Kolowy  Ob;
}
```

Wynik działania

```
++KolaPrzednie
++KolaTylnie
++Pojazd4Kolowy
```

Pola inicjalizowane są zgodnie z kolejnością ich deklaracji w klasie.

Konstruktory i destruktory w klasach złożonych

```
struct KolaPrzednie { //.....
    KolaPrzednie( )   { cout << "++KolaPrzednie" << endl; }
}; // ~KolaPrzednie( ) { cout << "--KolaPrzednie" << endl; }.....

struct KolaTylnie { //.....
    KolaTylnie( )   { cout << "++KolaTylnie" << endl; }
}; // ~KolaTylnie( ) { cout << "--KolaTylnie" << endl; }.....

struct Pojazd4Kolowy { //.....
    KolaPrzednie      _KolaPrz;
    KolaTylnie        _KolaTyl;

    Pojazd4Kolowy( )   { cout << "++Pojazd4Kolowy" << endl; }
}; // ~Pojazd4Kolowy( ) { cout << "--Pojazd4Kolowy" << endl; }.....

int main( )
{
    Pojazd4Kolowy  Ob;
    cout << "Witajcie i zegnajcie !!!" << endl;
}
```

Wynik działania

Konstruktory i destruktory w klasach złożonych

```
struct KolaPrzednie { //.....
    KolaPrzednie( )   { cout << "++KolaPrzednie" << endl; }
}; // ~KolaPrzednie( ) { cout << "--KolaPrzednie" << endl; }
.....

struct KolaTylnie { //.....
    KolaTylnie( )   { cout << "++KolaTylnie" << endl; }
}; // ~KolaTylnie( ) { cout << "--KolaTylnie" << endl; }
.....

struct Pojazd4Kolowy { //.....
    KolaPrzednie      _KolaPrz;
    KolaTylnie        _KolaTyl;

    Pojazd4Kolowy( )   { cout << "++Pojazd4Kolowy" << endl; }
}; // ~Pojazd4Kolowy( ) { cout << "--Pojazd4Kolowy" << endl; }
.....

int  main( )
{
    Pojazd4Kolowy  Ob;
    cout << "Witajcie i zegnajcie !!!" << endl;
}
```

Wynik działania

```
++KolaPrzednie
++KolaTylnie
++Pojazd4Kolowy
```

Konstruktory i destruktory w klasach złożonych

```
struct KolaPrzednie { //.....
    KolaPrzednie( )   { cout << "++KolaPrzednie" << endl; }
}; // ~KolaPrzednie( ) { cout << "--KolaPrzednie" << endl; }
.....

struct KolaTylnie { //.....
    KolaTylnie( )   { cout << "++KolaTylnie" << endl; }
}; // ~KolaTylnie( ) { cout << "--KolaTylnie" << endl; }
.....

struct Pojazd4Kolowy { //.....
    KolaPrzednie      _KolaPrz;
    KolaTylnie        _KolaTyl;

    Pojazd4Kolowy( )   { cout << "++Pojazd4Kolowy" << endl; }
}; // ~Pojazd4Kolowy( ) { cout << "--Pojazd4Kolowy" << endl; }
.....

int main( )
{
    Pojazd4Kolowy  Ob;
    cout << "Witajcie i zegnajcie !!!" << endl;
}
```

Wynik działania

```
++KolaPrzednie
++KolaTylnie
++Pojazd4Kolowy
"Witajcie i zegnajcie !!!"
```

Konstruktory i destruktory w klasach złożonych

```
struct KolaPrzednie { //.....
    KolaPrzednie( )   { cout << "++KolaPrzednie" << endl; }
}; // ~KolaPrzednie( ) { cout << "--KolaPrzednie" << endl; }.....

struct KolaTylnie { //.....
    KolaTylnie( )   { cout << "++KolaTylnie" << endl; }
}; // ~KolaTylnie( ) { cout << "--KolaTylnie" << endl; }.....

struct Pojazd4Kolowy { //.....
    KolaPrzednie      _KolaPrz;
    KolaTylnie        _KolaTyl;

    Pojazd4Kolowy( )   { cout << "++Pojazd4Kolowy" << endl; }
}; // ~Pojazd4Kolowy( ) { cout << "--Pojazd4Kolowy" << endl; }.....

int main( )
{
    Pojazd4Kolowy  Ob;
    cout << "Witajcie i zegnajcie !!!" << endl;
}
```

Wynik działania

```
++KolaPrzednie
++KolaTylnie
++Pojazd4Kolowy
"Witajcie i zegnajcie !!!"
--Pojazd4Kolowy
--KolaTylnie
--KolaPrzednie
```

Destrukcja następuje w kolejności odwrotnej.

Plan prezentacji

Dwa sposoby inicjalizacji

```
class Kolor { //.....
        int _Kod_Koloru;

    public :
        Kolor( );
};.....
```

Dwa sposoby inicjalizacji

```
class Kolor { //.....
        int  _Kod_Koloru;

    public :
        Kolor( );
};.....
```

```
Kolor::Kolor( )
 {
     _Kod_Koloru = 0;
 }
```

```
Kolor::Kolor( ): _Kod_Koloru(0)
 { }
```

Pole klasy może zostać zainicjalizowane na dwa sposoby, albo w ciele konstruktora, albo też poprzez listę inicjalizacyjną. Niektóre rodzaje pól mogą być zainicjalizowane tylko i wyłącznie z poziomu listy inicjalizacyjnej.

Lista inicjalizacyjna – różniece i podobieństwa

```
int   NrTeczy;
NrTeczy = 0;
```

```
Kolor::Kolor( )
 {
      _Kod_Koloru = 0;
 }
```

```
int   NrTeczy  =  0;
```

```
Kolor::Kolor( ): _Kod_Koloru(0)
 { }
```

Umieszczenie inicjalizatora w liście inicjalizacyjnej konstruktora powoduje przypisanie wartości polu w momencie jego utworzenia.

Lista inicjalizacyjna – parametryzacja

```
class Kolor { //.....
        int _Kod_Koloru;

    public :
        Kolor( int Param );
}; .....

Kolor::Kolor( int Kod ): _Kod_Koloru(Kod)
  { }
```

Do inicjalizatorów można przekazać wartość poprzez parametr wywołania konstruktora.

Lista inicjalizacyjna – parametryzacja

```
class PrzedzialLiczb { //.....
        int _WartoscMini;
        int _WartoscMaks;
    public :
        PrzedzialLiczb( int Mini, int Maks );
}; //.....

PrzedzialLiczb::PrzedzialLiczb( int Mini, int Maks ):
    _WartoscMini(Mini), _WartoscMaks(Maks)
{ }
```

Lista inicjalizacyjna może zawierać dowolną ilość inicjalizatorów. Uruchamiane są one w kolejności tworzenia pól, która jest zgodna z porządkiem ich deklaracji w klasie. Nie jest błędem podanie inicjalizatorów w innej kolejności. Jednak nie wpłynie ona na kolejność ich wywołań.
Aby nie wprowadzać nieporozumień, kolejność inicjalizatorów powinna zawsze odpowiadać faktycznej kolejności ich uruchamiania.

Wywoływanie konstruktorów w liście inicjalizacyjnej

```
struct Wektor { //.....
        float x, y;
        Wektor( ) { x = y = 0; }
        Wektor( float xx, float yy ) { x = xx; y = xy; }
}; //.....

struct Odcinek { //.....
      Wektor _Po, _Pn;
      Odcinek( );
}; //.....

Odcinek::Odcinek( ): _Po(-1,-1), _Pn(1,1) ..... Lista inicjalizująca wymusza wywołanie konstruktorów
{ }
```

Lista inicjalizująca pozwala wymusić wywołanie zadanego konstruktora dla poszczególnych składników obiektu. Kolejność składników listy powinna odpowiadać kolejności deklaracji pól klasy.

Plan prezentacji

Zmienne globalne

```
#include <stdio.h>

int IloscPakietow = 0;

void WyslijPakiet( )
{
    ...
    printf("Pakiet nr: %i\n",++IloscPakietow);
}

int main( )
{
    WyslijPakiet( );
    WyslijPakiet( );
    ...
    WyslijPakiet( );
    return 0;
}
```

Zmienne globalne

```
#include <stdio.h>

int lloscPakietow = 0;

void WyslijPakiet( )
{
    ...
    printf("Pakiet nr: %i\n",++lloscPakietow);
}

int main( )
{
    WyslijPakiet( );
    WyslijPakiet( );
    ...
    WyslijPakiet( );
    return 0;
}
```

```
Pakiet nr: 1
Pakiet nr: 2
Pakiet nr: 3
```

Zmienne globalne

```
#include <stdio.h>

int IloscPakietow = 0;

void WyslijPakiet( )
{
    ...
    printf("Pakiet nr: %i\n",++IloscPakietow);
}

int main( )
{
    WyslijPakiet( );
    WyslijPakiet( );
    ...
    WyslijPakiet( );
    return 0;
}
```

```
Pakiet nr: 1
Pakiet nr: 2
Pakiet nr: 11
```

Zmienne globalne

```
#include <stdio.h>

int IloscPakietow = 0;

void WyslijPakiet( )
{
    ...
    printf("Pakiet nr: %i\n",++IloscPakietow);
}

int main( )
{
    WyslijPakiet( );
    WyslijPakiet( );
    IloscPakietow = 10;
    WyslijPakiet( );
    return 0;
}
```

```
Pakiet nr: 1
Pakiet nr: 2
Pakiet nr: 11
```

Zmienne lokalne

```
#include <stdio.h>

void WyslijPakiet( )
{
    int IloscPakietow = 0;
    ...
    printf("Pakiet nr: %i\n",++IloscPakietow);
}

int main( )
{
    WyslijPakiet( );
    WyslijPakiet( );
    IloscPakietow = 10;
    WyslijPakiet( );
    return 0;
}
```

?

Zmienne lokalne

```
#include <stdio.h>

void WyslijPakiet( )
{
    int IloscPakietow = 0;
    ...
    printf("Pakiet nr: %i\n",++IloscPakietow);
}

int main( )
{
    WyslijPakiet( );
    WyslijPakiet( );
    IloscPakietow = 10;
    WyslijPakiet( );
    return 0;
}
```

```
Pakiet nr: 1
Pakiet nr: 1
Pakiet nr: 1
```

Zmienne statyczne

```
#include <stdio.h>

void WyslijPakiet( )
{
    static int IloscPakietow = 0;
    ...
    printf("Pakiet nr: %i\n",++IloscPakietow);
}

int main( )
{
    WyslijPakiet( );
    WyslijPakiet( );
    IloscPakietow = 10;
    WyslijPakiet( );
    return 0;
}
```

```
Pakiet nr: 1
Pakiet nr: 2
Pakiet nr: 3
```

Zmienne statyczne w języku *C* pełnią rolę zmiennych globalnych z ograniczonym zakresem dostępu tylko do poziomu funkcji, w której są definiowane.

Zmienne statyczne

```
#include <iostream>

void WyslijPakiet( )
{
    static int IloscPakietow = 0;
    ...
    std::cout << "Pakiet nr: " << ++IloscPakietow << std::endl;
}

int main( )
{
    WyslijPakiet( );
    WyslijPakiet( );
    IloscPakietow = 10;
    WyslijPakiet( );
}
```

```
Pakiet nr: 1
Pakiet nr: 2
Pakiet nr: 3
```

Identyczne znaczenie mają również w języku *C++* .

Plan prezentacji

Pola statyczne w klasie

```
class LaczeSieciowe {
    public :
      static int _IloscLaczy;

      LaczeSieciowe( ) { ++_IloscLaczy; }
      ~LaczeSieciowe( ) { --_IloscLaczy; }
};

int LaczeSieciowe ::_IloscLaczy = 0;

int main( )
{
    LaczeSieciowe Lacze1;

    cout << Lacze1._IloscLaczy << endl;
}
```

Pole statyczne pełni rolę zmiennej globalnej w ramach danej klasy. Istnieje tylko w jednym egzemplarzu niezależnie od wszystkich obiektów.

Pola statyczne w klasie

```
class LaczeSieciowe {
    public :
      static int _IloscLaczy;

      LaczeSieciowe( ) { ++_IloscLaczy; }
      ~LaczeSieciowe( ) { --_IloscLaczy; }
};

int LaczeSieciowe ::_IloscLaczy = 0;

int main( )
{
    LaczeSieciowe Lacze1;

    cout << Lacze1._IloscLaczy << endl;
}
```

Musi być zadeklarowane oraz zainicjalizowane poza klasą w przestrzeni zmiennych globalnych. Określony zostaje w ten sposób moment jego utworzenia.

Pola statyczne w klasie

```
class LaczeSieciowe {
    public :
      static int _IloscLaczy;

      LaczeSieciowe( ) { ++_IloscLaczy; }
      ~LaczeSieciowe( ) { --_IloscLaczy; }
};

int LaczeSieciowe ::_IloscLaczy;

int main( )
{
    LaczeSieciowe Lacze1;

    cout << Lacze1._IloscLaczy << endl;
}
```

Czy brak inicjalizacji oznacza wartość przypadkową?

Pola statyczne w klasie

```
class LaczeSieciowe {
    public :
      static int _IloscLaczy;

      LaczeSieciowe( ) { ++_IloscLaczy; }
      ~LaczeSieciowe( ) { --_IloscLaczy; }
};

int LaczeSieciowe ::_IloscLaczy;

int main( )
{
    LaczeSieciowe Lacze1;

    cout << Lacze1._IloscLaczy << endl;
}
```

Pola statyczne typów wbudowanych zawsze inicjalizowane są wartością 0.

Pola statyczne w klasie

```
class LaczeSieciowe {
    public :
      static int _IloscLaczy;

      LaczeSieciowe( ) { ++_IloscLaczy; }
      ~LaczeSieciowe( ) { --_IloscLaczy; }
};

int LaczeSieciowe ::_IloscLaczy;

int main( )
{

    cout << LaczeSieciowe::_IloscLaczy << endl;
}
```

Do pola statycznego można odwoływać się bez konieczności tworzenia obiektu, gdyż jest ono własnością klasy, a nie obiektu.

Pola statyczne w klasie

```
class LaczeSieciowe {
    public :
      static int _IloscLaczy;

      LaczeSieciowe( ) { ++_IloscLaczy; }
      ~LaczeSieciowe( ) { --_IloscLaczy; }
};

int LaczeSieciowe ::_IloscLaczy;

int main( )
{
    LaczeSieciowe Lacze1, Lacze2;

    cout << LaczeSieciowe::_IloscLaczy << endl;
     cout << Lacze1._IloscLaczy << endl;
     cout << Lacze2._IloscLaczy << endl;
}
```

Dlatego też każdy z tych trzech sposobów odwołania się do pola `_IloscLaczy` jest odwołaniem się do tego samego obszaru pamięci.

Pola statyczne w klasie

```
class LaczeSieciowe {
    public :
      static int _IloscLaczy;

      LaczeSieciowe( ) { ++_IloscLaczy; }
      ~LaczeSieciowe( ) { --_IloscLaczy; }
};

int LaczeSieciowe ::_IloscLaczy;

int main( )
{

    cout << LaczeSieciowe::_IloscLaczy << endl;


}
```

Aby móc odwołać się do pola statycznego, nie musi istnieć żaden obiekt. Pola statyczne istnieją niezależnie od poszczególnych obiektów.

Pola statyczne w klasie

```
class LaczeSieciowe {
    public :
      static int _IloscLaczy;

      LaczeSieciowe( ) { ++_IloscLaczy; }
      ~LaczeSieciowe( ) { --_IloscLaczy; }
};

int LaczeSieciowe ::_IloscLaczy;

int main( )
{

    LaczeSieciowe::_IloscLaczy += 1000000;


}
```

W tym konkretnym rozwiązaniu definiowanie zmiennej statycznej w sekcji publicznej nie jest dobrym rozwiązaniem.

Plan prezentacji

Statyczne metody klasy

```
class LaczeSieciowe {
      static int _IloscLaczy;

    public :
      LaczeSieciowe( ) { ++_IloscLaczy; }
      ~LaczeSieciowe( ) { --_IloscLaczy; }
};

int LaczeSieciowe ::_IloscLaczy = 0;

int main( )
{
  LaczeSieciowe Lacze1;

  cout << Lacze1._IloscLaczy << endl;

}
```

Bezwzględu na to, czy pole statyczne zdefiniowane jest sekcji publicznej, czy też prywatnej, należy je zainicjalizować w miejscu definicji. Nie ma tutaj żadnego ograniczenia dostępu.

Statyczne metody klasy

```
class LaczeSieciowe {
        static  int  _IloscLaczy;

      public :
        LaczeSieciowe( )  { ++_IloscLaczy; }
        ~LaczeSieciowe( )  { --_IloscLaczy; }
};

int  LaczeSieciowe ::_IloscLaczy = 0;

int  main( )
{
  LaczeSieciowe  Lacze1;

  cout << Lacze1._IloscLaczy << endl;

}
```

Dla prób odwoływania się do prywatnego pola statycznego poza jego definicją obowiązują takie same reguły dostępu jak dla każdego innego pola.

Statyczne metody klasy

```
class LaczeSieciowe {
        static  int  _IloscLaczy;

      public :
        LaczeSieciowe( )  { ++_IloscLaczy; }
        ~LaczeSieciowe( )  { --_IloscLaczy; }
};

int  LaczeSieciowe ::_IloscLaczy = 0;

int  main( )
{
  LaczeSieciowe  Lacze1;

  cout << Lacze1._IloscLaczy << endl;
  cout << LaczeSieciowe::_IloscLaczy << endl;
}
```

Dotyczy to również prób odwołania się do takiego pola bez pośrednictwa obiektu.

Statyczne metody klasy

```
class LaczeSieciowe {
        static  int  _IloscLaczy;

      public :
        LaczeSieciowe( )  { ++_IloscLaczy; }
        ~LaczeSieciowe( )  { --_IloscLaczy; }
       int WezIloscLaczy( ) const { return  _IloscLaczy; }
};

 ...
int  main( )
{
  LaczeSieciowe  Lacze1;

  cout << Lacze1.WezIloscLaczy( ) << endl;
  cout << LaczeSieciowe::_IloscLaczy << endl;
}
```

Można zdefiniować jednak zwykłą metodę, która udostępni wartość tego pola. Wadą tego rozwiązania jest to, że taką metodę można wywołać tylko za pośrednictwem obiektu danej klasy.

Statyczne metody klasy

```
class LaczeSieciowe {
        static  int  _IloscLaczy;

      public :
        LaczeSieciowe( )  { ++_IloscLaczy; }
        ~LaczeSieciowe( )  { --_IloscLaczy; }
        static  int WezIloscLaczy( )  { return  _IloscLaczy; }
};

 ...
int  main( )
{
  LaczeSieciowe  Lacze1;

  cout << Lacze1.WezIloscLaczy( ) << endl;
  cout << LaczeSieciowe::WezIloscLaczy( ) << endl;
}
```

Tej wady nie mają metody statyczne.
Uwaga: *Metody statyczne nie mogą być metodami typu* **const**. *Dlaczego?*

Plan prezentacji

Rozmiar obiektów statycznych

```
class LaczeSieciowe {
        static  int  _IloscLaczy;
        int          _ID;

      public :
        LaczeSieciowe( )  { ++_IloscLaczy; }
        ~LaczeSieciowe( )  { --_IloscLaczy; }
};

int  main( )
{
  cout << sizeof (int) << endl;
  cout << sizeof (LaczeSieciowe) << endl;
}
```

Wynik działania:

Rozmiar obiektów statycznych

```
class LaczeSieciowe {
        static  int  _IloscLaczy;
        int          _ID;

      public :
        LaczeSieciowe( )  { ++_IloscLaczy; }
        ~LaczeSieciowe( )  { --_IloscLaczy; }
};

int  main( )
{
  cout << sizeof (int) << endl;
  cout << sizeof (LaczeSieciowe) << endl;
}
```

Wynik działania:

```
4
```

Rozmiar obiektów statycznych

```
class LaczeSieciowe {
        static  int  _IloscLaczy;
        int          _ID;

      public :
        LaczeSieciowe( )  { ++_IloscLaczy; }
        ~LaczeSieciowe( )  { --_IloscLaczy; }
};

int  main( )
{
  cout << sizeof (int) << endl;
  cout << sizeof (LaczeSieciowe) << endl;
}
```

Wynik działania:

```
4
```

Rozmiar obiektów statycznych

```
class LaczeSieciowe {
        static  int  _IloscLaczy;
        int          _ID;

      public :
        LaczeSieciowe( )  { ++_IloscLaczy; }
        ~LaczeSieciowe( )  { --_IloscLaczy; }
};

int  main( )
{
  cout << sizeof (int) << endl;
  cout << sizeof (LaczeSieciowe) << endl;
}
```

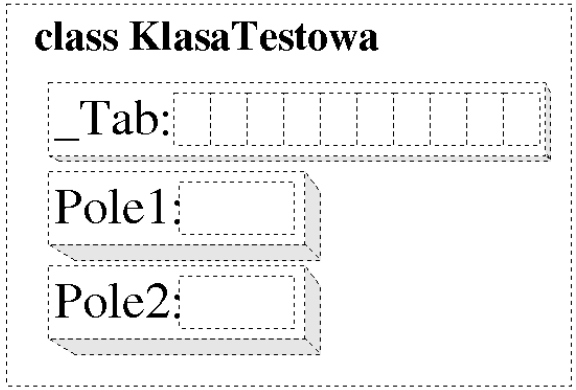
Wynik działania:

```
4
4
```

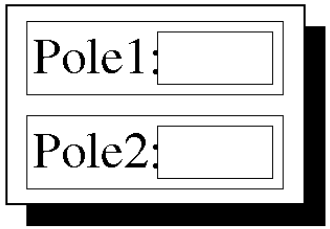
Rozmiar obiektów statycznych

```
class KlasaTestowa {
      static   int   _Tab[10];
      int            Pole1;
      int            Pole2;
};

int  KlasaTestowa::_Tab[10];

int  main( )
{
   KlasaTestowa    ObTestowy;

}
```

Rozmiar obiektów statycznych




```
int main()
{
 KlasaTestowa
}
```



Plan prezentacji

Kilka prostych pojęć

Figura geometryczna, *Okrąg,* *Kwadrat*

Wzajemne zależności

Figura geometryczna

Okrąg

Kwadrat

Wzajemne zależności

Figura geometryczna

Okrąg

– jest *figurą geometryczną* i nie jest *kwadratem*

Kwadrat

Wzajemne zależności

Figura geometryczna

Okrąg – jest *figurą geometryczną* i nie jest *kwadratem*

Kwadrat – jest *figurą geometryczną*, i nie jest *okręgiem*

Wzajemne zależności

Figura geometryczna – może być *kwadratem* lub *okręgiem*

Okrąg – jest *figurą geometryczną* i nie jest *kwadratem*

Kwadrat – jest *figurą geometryczną*, i nie jest *okręgiem*

Charakterystyczne cechy

Figura geometryczna

Okrąg

Kwadrat

Charakterystyczne cechy

Figura geometryczna

- pole powierzchni,
- obwód.

Okrąg

Kwadrat

Charakterystyczne cechy

Figura geometryczna

- pole powierzchni,
- obwód.

Okrąg

Jest figurą geometryczną

- pole powierzchni,
- obwód.

Cechy szczególne

- długość promienia.

Kwadrat

Charakterystyczne cechy

Figura geometryczna

- pole powierzchni,
- obwód.

Okrąg

Jest figurą geometryczną

- pole powierzchni,
- obwód.

Cechy szczególne

- długość promienia.

Kwadrat

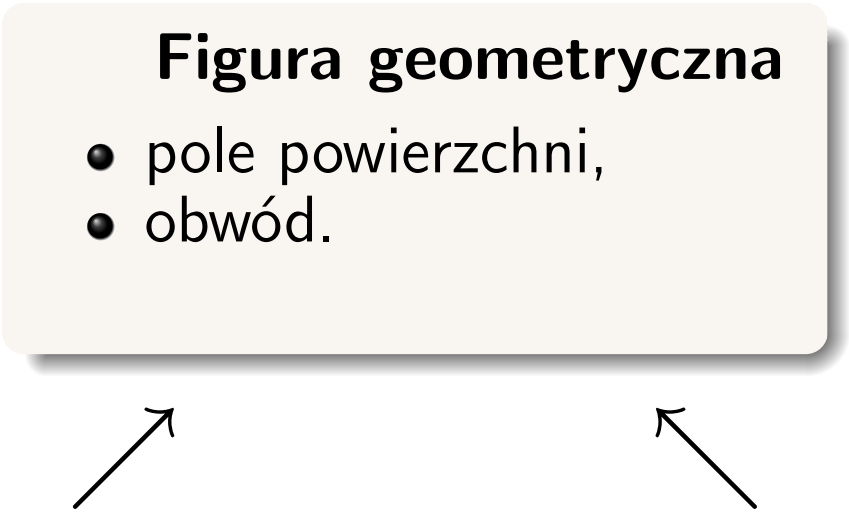
Jest figurą geometryczną

- pole powierzchni,
- obwód.

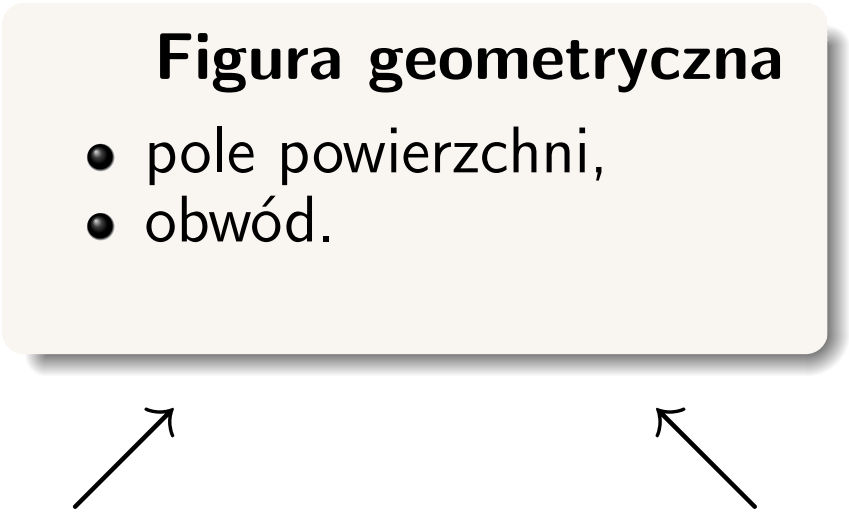
Cechy szczególne

- długość boku.

Dziedziczenie



Dziedziczenie



Okrąg

Jest figurą geometryczną
- pole powierzchni,
- obwód.

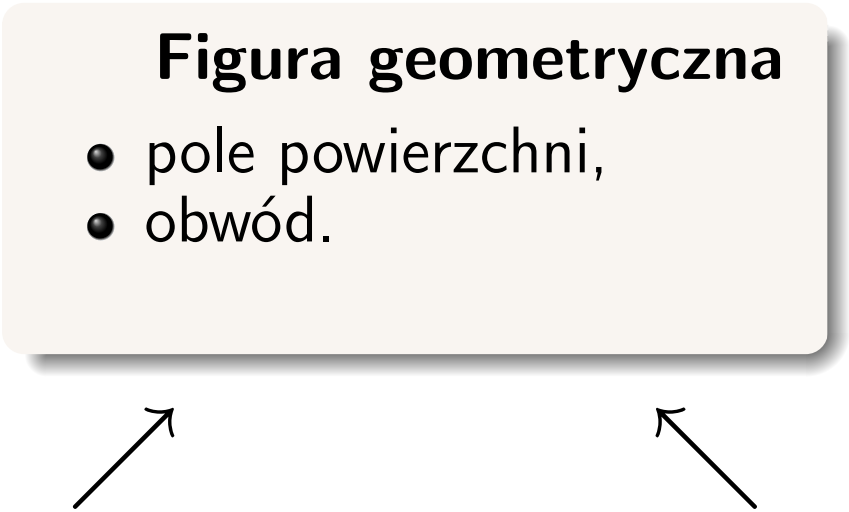
Cechy szczególne
- długość promienia.

Kwadrat

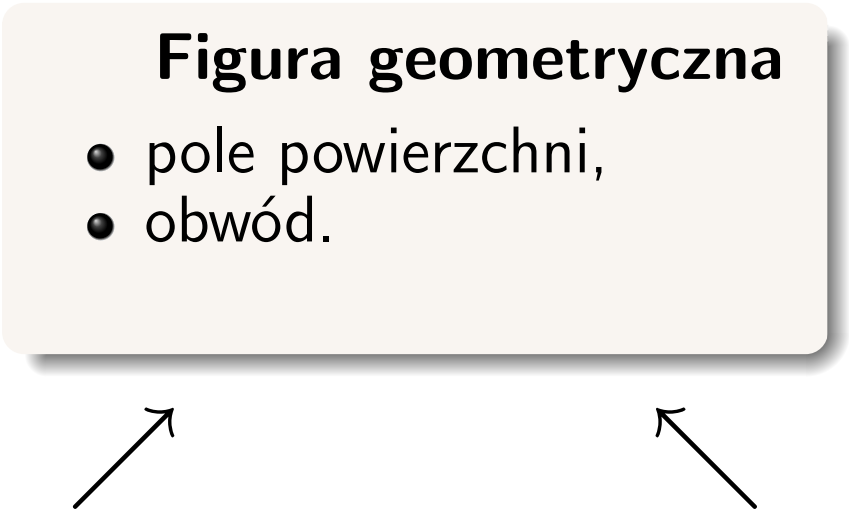
Jest figurą geometryczną
- pole powierzchni,
- obwód.

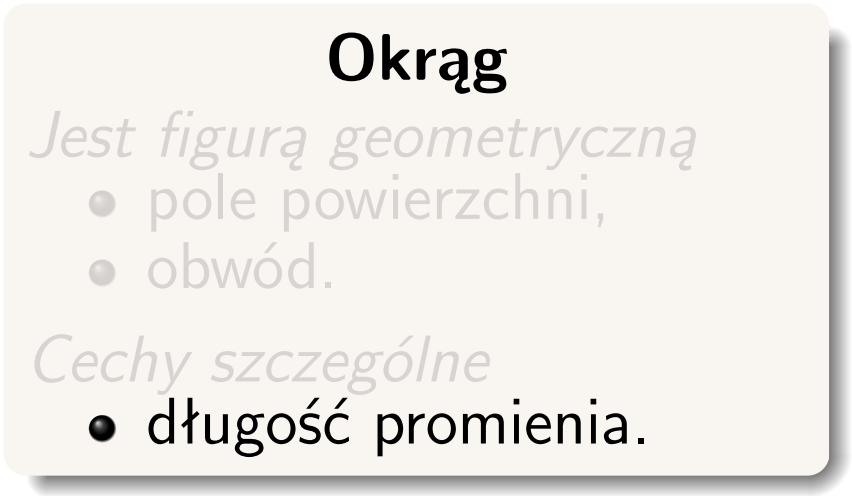
Cechy szczególne
- długość boku.

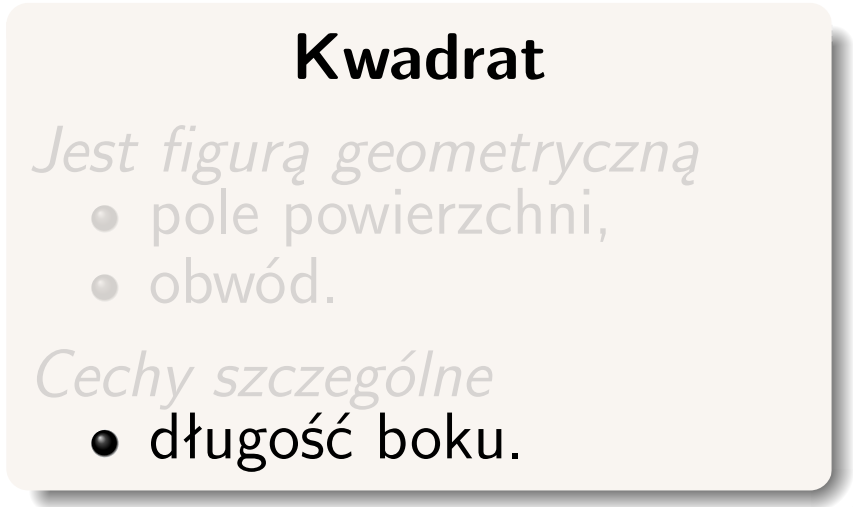
Dziedziczenie



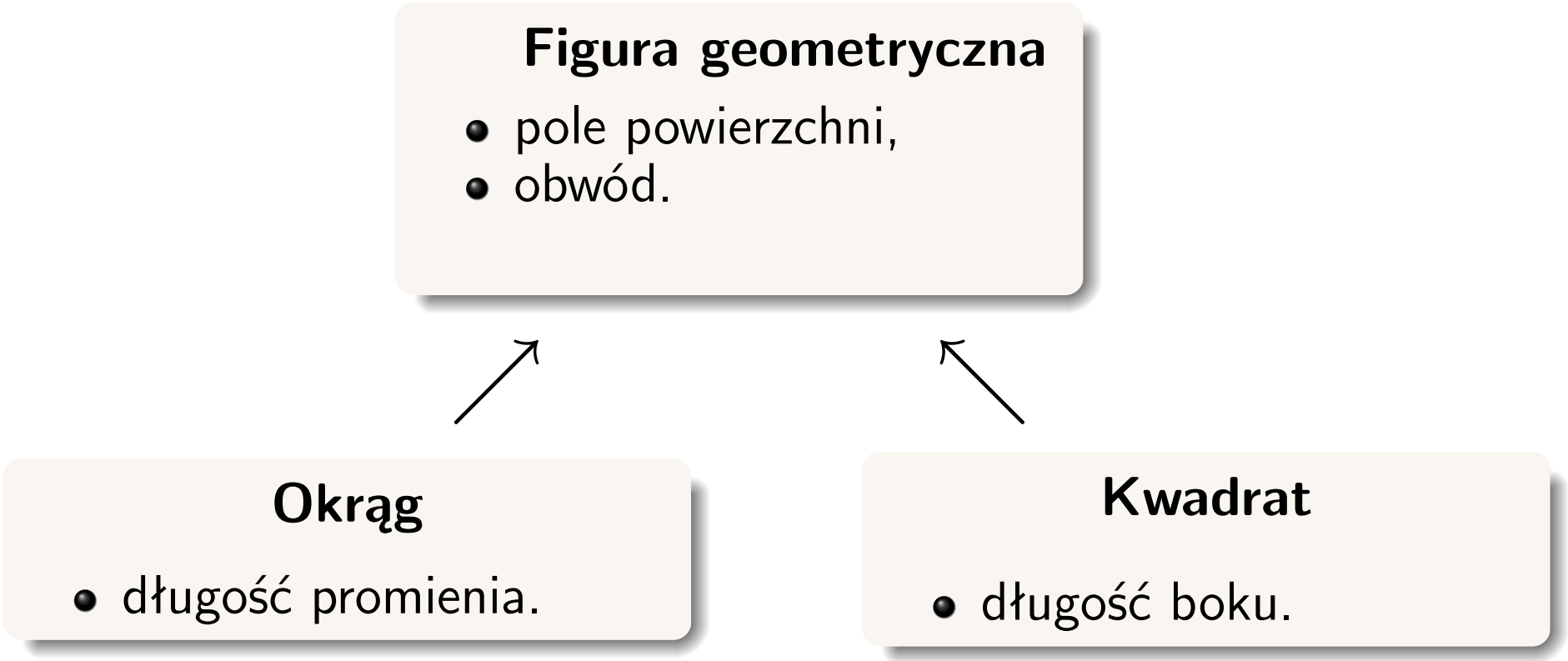
Dziedziczenie







Dziedziczenie



Dziedziczenie

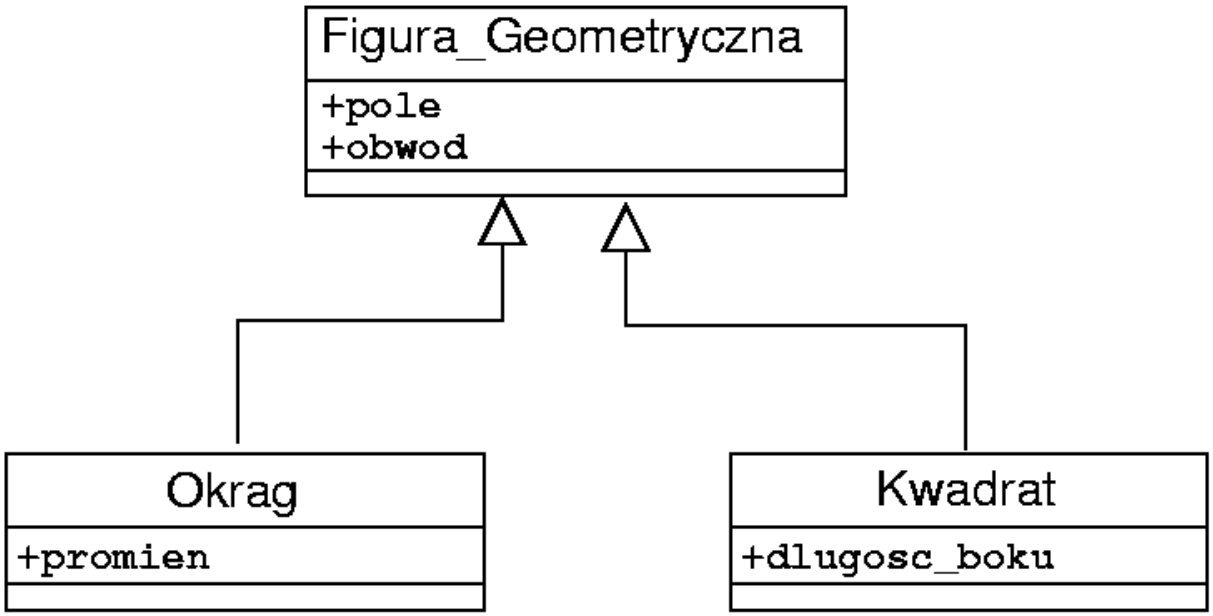


Diagram klas w języku UML odzwierciedlający fakt dziedziczenia klasy `Figura_Geometryczna` przez klasy `Okrąg` i `Kwadrat`.

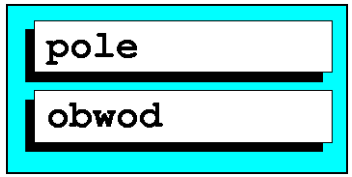
Plan prezentacji

Zapis w języku C++

```
struct Figura_Geometryczna { // .....
    double pole;
    double obwod;
}; // .....

struct Okrag: Figura_Geometryczna { // .....
    double promien;
}; // .....

struct Kwadrat: Figura_Geometryczna { // .....
    double dlugosc_boku;
}; // .....
```

Zapis w języku C++

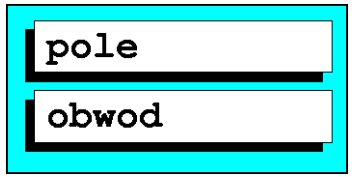
```
struct Figura_Geometryczna { // ..........
    double pole;
    double obwod;
}; // ..........
```



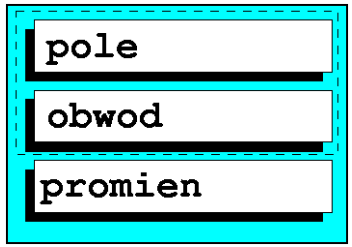

```
struct Okrag: Figura_Geometryczna { // ..........
    double promien;
}; // ..........

struct Kwadrat: Figura_Geometryczna { // ..........
    double dlugosc_boku;
}; // ..........
```

Zapis w języku C++

```
struct Figura_Geometryczna { // ..........
    double pole;
    double obwod;
}; // ..........
```



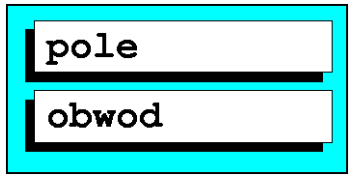

```
struct Okrag: Figura_Geometryczna { // ..........
    double promien;
}; // ..........
```



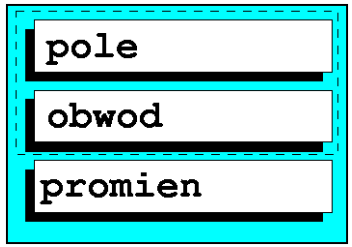

```
struct Kwadrat: Figura_Geometryczna { // ..........
    double dlugosc_boku;
}; // ..........
```

Zapis w języku C++

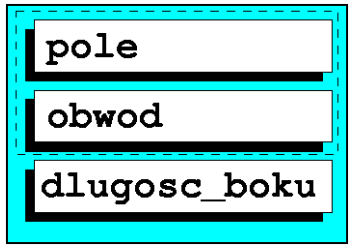
```
struct Figura_Geometryczna { // .....
    double pole;
    double obwod;
}; // .....
```




```
struct Okrag: Figura_Geometryczna { // .....
    double promien;
}; // .....
```

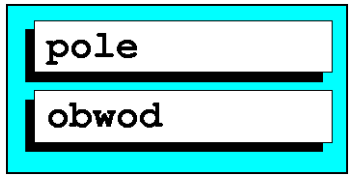



```
struct Kwadrat: Figura_Geometryczna { // .....
    double dlugosc_boku;
}; // .....
```

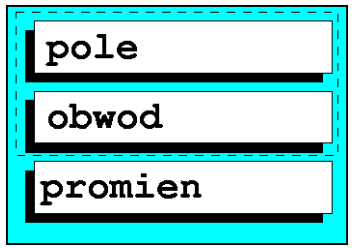


Równoważny zapis bez dziedziczenia

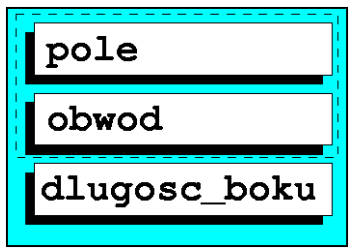
```
struct Figura_Geometryczna { //
    double pole;
    double obwod;
}; //
```




```
struct Okrag: Figura_Geometryczna { //
    double pole;
    double obwod;
    double promien;
}; //
```




```
struct Kwadrat { //
    double pole;
    double obwod;
    double dlugosc_boku;
}; //
```



- Dziedziczenie pozwala na odzwierciedlenie hierarchii modelowanych pojęć i związków między nimi.
- Dodatkową istotną korzyścią jest, to że możemy w prosty sposób wykorzystać kod dziedziczonej klasy bazowej w klasie pochodnej.
  Bez dziedzicznie konieczne byłoby napisanie wielokrotnie tego samego kodu w klasach pochodnych.

Plan prezentacji

Tryby dziedziczenia

Tryby dziedziczenia:

- publiczny $\longrightarrow$ **class** Klasa_Pochodna: **public** Klasa_Bazowa { ... };
- chroniony $\longrightarrow$ **class** Klasa_Pochodna: **protected** Klasa_Bazowa { ... };
- prywatny $\longrightarrow$ **class** Klasa_Pochodna: **private** Klasa_Bazowa { ... };

Tryby dziedziczenia

Tryby dziedziczenia:

- publiczny $\longrightarrow$ **class** Klasa_Pochodna: **public** Klasa_Bazowa { ... };
- chroniony $\longrightarrow$ **class** Klasa_Pochodna: **protected** Klasa_Bazowa { ... };
- prywatny $\longrightarrow$ **class** Klasa_Pochodna: **private** Klasa_Bazowa { ... };

class Klasa_Pochodna: Klasa_Bazowa ... $\longrightarrow$ **class** Klasa_Pochodna: **private** Klasa_Bazowa ...

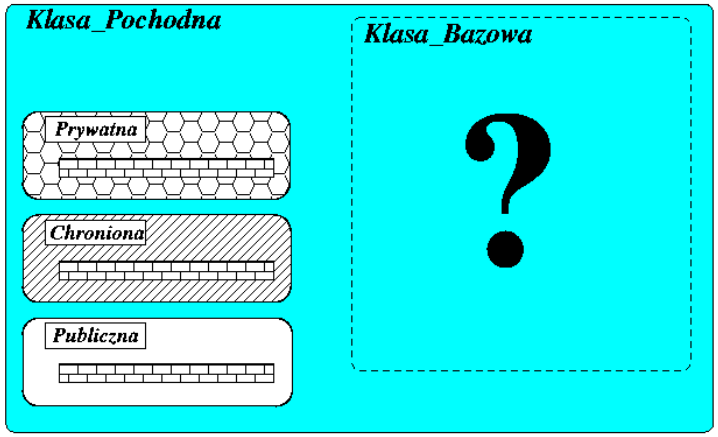
struct Klasa_Pochodna: Klasa_Bazowa ... $\longrightarrow$ **struct** Klasa_Pochodna: **public** Klasa_Bazowa ...

Tryby dziedziczenia

Tryby dziedziczenia:

- publiczny $\longrightarrow$ **class** Klasa_Pochodna: **public** Klasa_Bazowa { ... };
- chroniony $\longrightarrow$ **class** Klasa_Pochodna: **protected** Klasa_Bazowa { ... };
- prywatny $\longrightarrow$ **class** Klasa_Pochodna: **private** Klasa_Bazowa { ... };

Klasa_Pochodna:
. . . Klasa_Bazowa
$\longrightarrow$



class Klasa_Pochodna: Klasa_Bazowa . . . $\longrightarrow$ **class** Klasa_Pochodna: **private** Klasa_Bazowa . . .

struct Klasa_Pochodna: Klasa_Bazowa . . . $\longrightarrow$ **struct** Klasa_Pochodna: **public** Klasa_Bazowa . . .

Tryby dziedziczenia

Tryby dziedziczenia:

- publiczny $\longrightarrow$ **class** Klasa_Pochodna: **public** Klasa_Bazowa { ... };
- chroniony $\longrightarrow$ **class** Klasa_Pochodna: **protected** Klasa_Bazowa { ... };
- prywatny $\longrightarrow$ **class** Klasa_Pochodna: **private** Klasa_Bazowa { ... };

Tryby dziedziczenia

Tryby dziedziczenia:

- publiczny $\longrightarrow$ **class** Klasa_Pochodna: **public** Klasa_Bazowa { ... };
- chroniony $\longrightarrow$ **class** Klasa_Pochodna: **protected** Klasa_Bazowa { ... };
- prywatny $\longrightarrow$ **class** Klasa_Pochodna: **private** Klasa_Bazowa { ... };

class Klasa_Pochodna: Klasa_Bazowa ... $\longrightarrow$ **class** Klasa_Pochodna: **private** Klasa_Bazowa ...

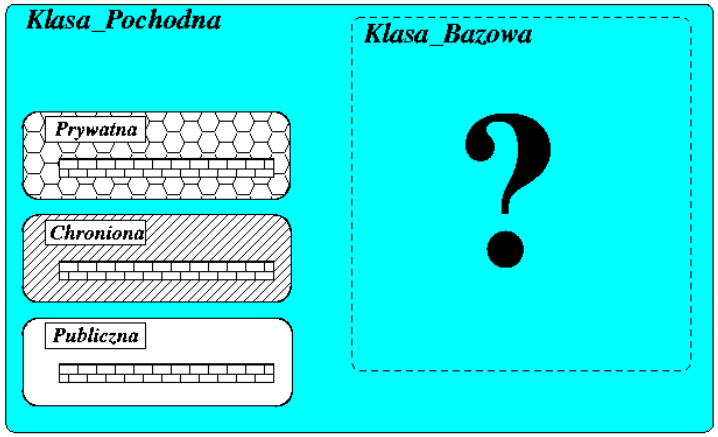
struct Klasa_Pochodna: Klasa_Bazowa ... $\longrightarrow$ **struct** Klasa_Pochodna: **public** Klasa_Bazowa ...

Tryby dziedziczenia

Tryby dziedziczenia:

- publiczny $\longrightarrow$ **class** Klasa_Pochodna: **public** Klasa_Bazowa { ... };
- chroniony $\longrightarrow$ **class** Klasa_Pochodna: **protected** Klasa_Bazowa { ... };
- prywatny $\longrightarrow$ **class** Klasa_Pochodna: **private** Klasa_Bazowa { ... };

Klasa_Pochodna:
. . . Klasa_Bazowa
$\longrightarrow$



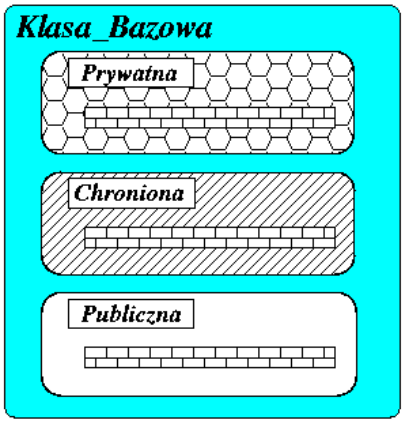
class Klasa_Pochodna: Klasa_Bazowa . . . $\longrightarrow$ **class** Klasa_Pochodna: **private** Klasa_Bazowa . . .

struct Klasa_Pochodna: Klasa_Bazowa . . . $\longrightarrow$ **struct** Klasa_Pochodna: **public** Klasa_Bazowa . . .

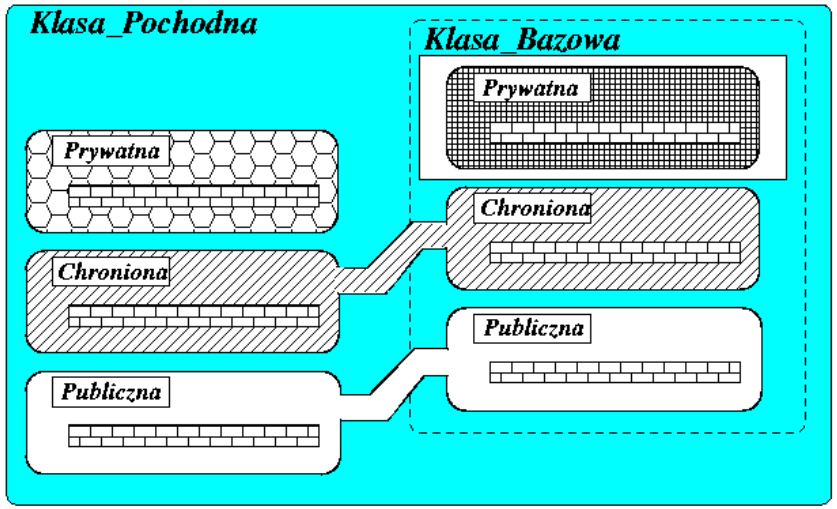
Dziedziczenie w trybie publicznym

Tryb dziedziczenia:

- publiczny $\longrightarrow$ **class** Klasa_Pochodna: **public** Klasa_Bazowa
  {
  . . .
  };



Klasa_Pochodna:
public Klasa_Bazowa
$\longrightarrow$



Dostęp do pól w klasie pochodnej

```
class KlasaBazowa {   //.....
      int  _PolePryw_KBaz;

    protected :
      int  _PoleChro_KBaz;

    public :
      int  _PolePubl_KBaz;
};   //.....

class KlasaPochodna: public  KlasaBazowa {   //.....
    public :
      void  OdwolanieDoPol( );
};   //.....

void  KlasaPochodna::OdwolanieDoPol( )
{
    _PolePubl_KBaz = _PoleChro_KBaz = _PolePryw_KBaz = 0;
}
```

Dostęp do pól w klasie pochodnej

```
class KlasaBazowa {   //.....
      int  _PolePryw_KBaz;

    protected :
      int  _PoleChro_KBaz;

    public :
      int  _PolePubl_KBaz;
};   //.....

class KlasaPochodna: public  KlasaBazowa {   //.....
    public :
      void  OdwolanieDoPol( );
};   //.....

void  KlasaPochodna::OdwolanieDoPol( )
{
    _PolePubl_KBaz = _PoleChro_KBaz =  _PolePryw_KBaz= 0;
}
```

Pola i metody prywatne klasy bazowej są niedostępne na poziomie metod klasy pochodnej. Dostępne są natomiast pola i metody znajdujące się w sekcji chronionej i publicznej. Własność ta nie zależy od sposobu dziedziczenia klasy bazowej.

Dostęp do pól obiektu

```
class KlasaBazowa {   // .....
        int  _PolePryw_KBaz;
      protected :
        int  _PoleChro_KBaz;
      public :
        int  _PolePubl_KBaz;
};   // .....

class KlasaPochodna: public  KlasaBazowa { . . . } // .....
```

```
int  main( )
{
   KlasaBazowa      ObBa;
   ObBa._PolePryw_KBaz = 1;
   ObBa._PoleChro_KBaz = 2;
   ObBa._PolePubl_KBaz = 3;
}
```

```
int  main( )
{
   KlasaPochodna    ObPo;
   ObPo._PolePryw_KBaz = 1;
   ObPo._PoleChro_KBaz = 2;
   ObPo._PolePubl_KBaz = 3;
}
```

Dostęp do pól obiektu

```
class KlasaBazowa {   // ..........
      int  _PolePryw_KBaz;
    protected :
      int  _PoleChro_KBaz;
    public :
      int  _PolePubl_KBaz;
};   // ..........

class KlasaPochodna: public KlasaBazowa { . . . } // ..........
```

W przypadku dziedziczenia w trybie publicznym pola i metody klasy bazowej, które są niedostępne poza tą klasą. Są również niedostępne w przypadku obiektu klasy pochodnej.

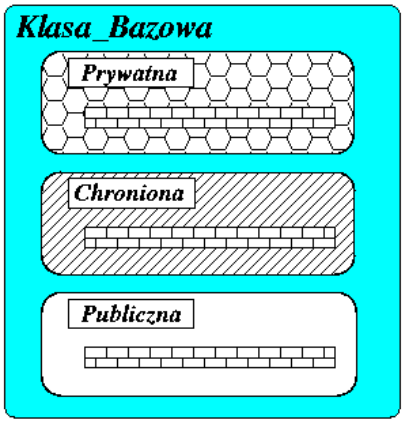
```
int  main( )
{
   KlasaBazowa      ObBa;
   ObBa._PolePryw_KBaz = 1;
   ObBa._PoleChro_KBaz = 2;
   ObBa._PolePubl_KBaz = 3;
}
```

```
int  main( )
{
   KlasaPochodna    ObPo;
   ObPo._PolePryw_KBaz = 1;
   ObPo._PoleChro_KBaz = 2;
   ObPo._PolePubl_KBaz = 3;
}
```

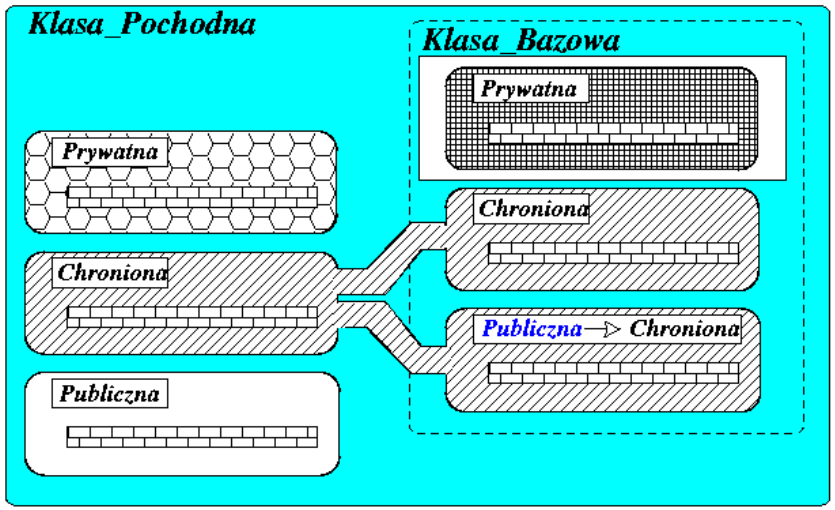
Dziedziczenie w trybie chronionym

Tryb dziedziczenia:

- chroniony $\longrightarrow$ **class** Klasa_Pochodna: **protected** Klasa_Bazowa
  {
  . . .
  };



Klasa_Pochodna:
protected Klasa_Bazowa
$\longrightarrow$



Dostęp do pól w klasie pochodnej

```
class KlasaBazowa {   //.....
      int  _PolePryw_KBaz;

    protected :
      int  _PoleChro_KBaz;

    public :
      int  _PolePubl_KBaz;
};   //.....

class KlasaPochodna: protected  KlasaBazowa {   //.....
    public :
      void  OdwolanieDoPol( );
};   //.....

void  KlasaPochodna::OdwolanieDoPol( )
{
    _PolePubl_KBaz = _PoleChro_KBaz = _PolePryw_KBaz = 0;
}
```

Dostęp do pól w klasie pochodnej

```
class KlasaBazowa {   //.....
      int  _PolePryw_KBaz;

    protected :
      int  _PoleChro_KBaz;

    public :
      int  _PolePubl_KBaz;
};   //.....

class KlasaPochodna: protected  KlasaBazowa {   //.....
    public :
      void  OdwolanieDoPol( );
};   //.....

void  KlasaPochodna::OdwolanieDoPol( )
{
    _PolePubl_KBaz = _PoleChro_KBaz =  _PolePryw_KBaz= 0;
}
```

Dziedziczenie w trybie chronionym nie ogranicza dostępności pól i metod klasy bazowej na poziomie metod klasy pochodnej. Nie powoduje też, że pola i metody prywatne mogłyby stać się dostępne dla klasy pochodnej.

Dostęp do pól obiektu

```
class KlasaBazowa {   // ..........................................
        int  _PolePryw_KBaz;
      protected :
        int  _PoleChro_KBaz;
      public :
        int  _PolePubl_KBaz;
};   // ..........................................

class KlasaPochodna: protected  KlasaBazowa { . . . } // ..................
```

```
int  main( )
{
   KlasaBazowa      ObBa;

   ObBa._PolePryw_KBaz = 1;
   ObBa._PoleChro_KBaz = 2;
   ObBa._PolePubl_KBaz = 3;
}
```

```
int  main( )
{
   KlasaPochodna    ObPo;

   ObPo._PolePryw_KBaz = 1;
   ObPo._PoleChro_KBaz = 2;
   ObPo._PolePubl_KBaz = 3;
}
```

Dostęp do pól obiektu

```
class KlasaBazowa {   // ..........................................
        int  _PolePryw_KBaz;
      protected :
        int  _PoleChro_KBaz;
      public :
        int  _PolePubl_KBaz;
};   // ..........................................
```

Dziedziczenie w trybie chronionym sprawia, że wszystkie komponenty klasy bazowej stają się niedostępne poza klasą pochodną.

```
class KlasaPochodna: protected  KlasaBazowa { . . . } // ..................
```

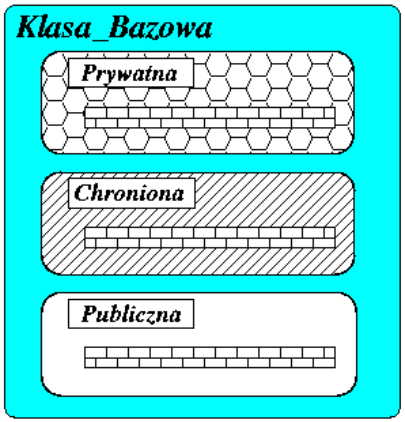
```
int  main( )
{
   KlasaBazowa      ObBa;
   ObBa._PolePryw_KBaz = 1;
   ObBa._PoleChro_KBaz = 2;
   ObBa._PolePubl_KBaz = 3;
}
```

```
int  main( )
{
   KlasaPochodna    ObPo;
   ObPo._PolePryw_KBaz = 1;
   ObPo._PoleChro_KBaz = 2;
   ObPo._PolePubl_KBaz = 3;
}
```

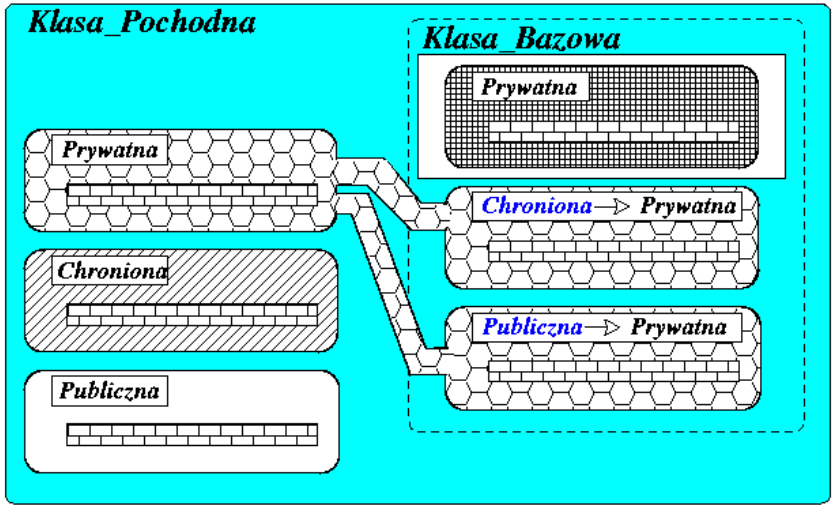
Dziedziczenie w trybie prywatnym

Tryb dziedziczenia:

- prywatnym $\longrightarrow$ **class** Klasa_Pochodna: **private** Klasa_Bazowa
  {
  . . .
  };



Klasa_Pochodna:
private Klasa_Bazowa
$\longrightarrow$



Dostęp do pól w klasie pochodnej

```
class KlasaBazowa {   //.....
      int  _PolePryw_KBaz;

    protected :
      int  _PoleChro_KBaz;

    public :
      int  _PolePubl_KBaz;
};   //.....

class KlasaPochodna: private  KlasaBazowa {   //.....
    public :
      void  OdwolanieDoPol( );
};   //.....

void  KlasaPochodna::OdwolanieDoPol( )
{
    _PolePubl_KBaz = _PoleChro_KBaz = _PolePryw_KBaz = 0;
}
```

Dostęp do pól w klasie pochodnej

```
class KlasaBazowa {   //.....
      int  _PolePryw_KBaz;

    protected :
      int  _PoleChro_KBaz;

    public :
      int  _PolePubl_KBaz;
};   //.....

class KlasaPochodna: private  KlasaBazowa {   //.....
    public :
      void  OdwolanieDoPol( );
};   //.....

void  KlasaPochodna::OdwolanieDoPol( )
{
    _PolePubl_KBaz = _PoleChro_KBaz =  _PolePryw_KBaz= 0;
}
```

Dziedziczenie w trybie prywatnym, podobnie jak dziedziczenie i w trybach publicznym i chronionym, nie ogranicza dostępności pól i metod klasy bazowej na poziomie metod klasy pochodnej. Nie powoduje też, że pola i metody prywatne mogłyby stać się dostępne dla klasy pochodnej.

Dostęp do pól obiektu

```
class KlasaBazowa {   // ..........................................
        int  _PolePryw_KBaz;
      protected :
        int  _PoleChro_KBaz;
      public :
        int  _PolePubl_KBaz;
};   // ..........................................

class KlasaPochodna: private  KlasaBazowa { . . . } // ..................
```

```
int  main( )
{
   KlasaBazowa      ObBa;

   ObBa._PolePryw_KBaz = 1;
   ObBa._PoleChro_KBaz = 2;
   ObBa._PolePubl_KBaz = 3;
}
```

```
int  main( )
{
   KlasaPochodna    ObPo;

   ObPo._PolePryw_KBaz = 1;
   ObPo._PoleChro_KBaz = 2;
   ObPo._PolePubl_KBaz = 3;
}
```

Dostęp do pól obiektu

```
class KlasaBazowa {   // ..........................................
        int  _PolePryw_KBaz;
      protected :
        int  _PoleChro_KBaz;
      public :
        int  _PolePubl_KBaz;
};   // ..........................................................

class KlasaPochodna: private  KlasaBazowa { . . . } // ..................
```

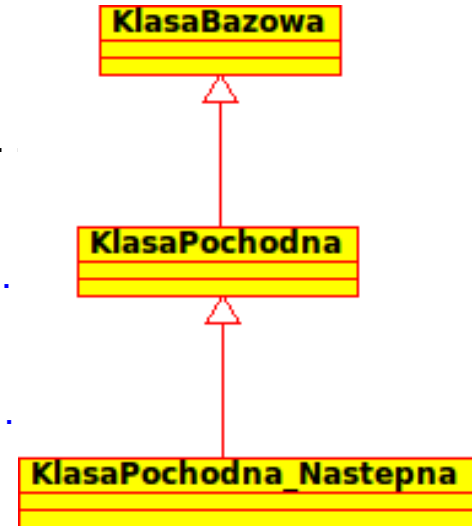
Dziedziczenie w trybie prywatnym, podobnie jak dziedziczenie w trybie chronionym, sprawia, że wszystkie komponenty klasy bazowej stają się niedostępne poza klasą pochodną.

```
int  main( )
{
   KlasaBazowa      ObBa;
   ObBa._PolePryw_KBaz = 1;
   ObBa._PoleChro_KBaz = 2;
   ObBa._PolePubl_KBaz = 3;
}
```

```
int  main( )
{
   KlasaPochodna    ObPo;
   ObPo._PolePryw_KBaz = 1;
   ObPo._PoleChro_KBaz = 2;
   ObPo._PolePubl_KBaz = 3;
}
```

Dostępność komponentów – dziedziczenie: tryp chroniony

```
class KlasaBazowa { // .....
        int _PolePryw_KBaz;
    protected :
        int _PoleChro_KBaz;
    public :
        int _PolePubl_KBaz;
}; // .....

class KlasaPochodna: protected KlasaBazowa { } // .....

class KlasaPochodna_Nastepna: public KlasaPochodna { // .....
    public :
        void OdwolanieDoPol( );
}; // .....
```




```
void KlasaPochodna_Nastepna::OdwolanieDoPol( )
{
    _PolePubl_KBaz = _PoleChro_KBaz = _PolePryw_KBaz = 0;
}
```

Dostępność komponentów – dziedziczenie: tryp chroniony

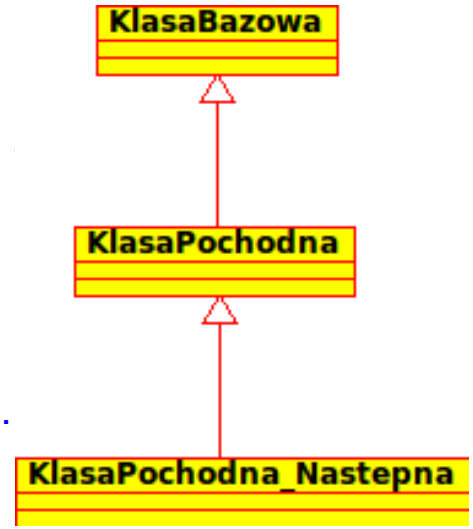
```
class KlasaBazowa { // .....
        int _PolePryw_KBaz;
    protected :
        int _PoleChro_KBaz;
    public :
        int _PolePubl_KBaz;
}; // .....

class KlasaPochodna: protected KlasaBazowa { } // .....

class KlasaPochodna_Nastepna: public KlasaPochodna { // .....
    public :
        void OdwolanieDoPol( );
}; // .....
```

Jeżeli klasa bazowa jest dziedziczona w trybie chronionym, to jej komponenty, które dostępne są w klasie pochodnej, będą dostępne również w klasie dziedziczącej klasę pochodną, zaś komponenty prywatne nie będą dostępne (niezależnie do trybu dziedziczenia).


```
void KlasaPochodna_Nastepna::OdwolanieDoPol( )
{
    _PolePubl_KBaz = _PoleChro_KBaz = _PolePryw_KBaz = 0;
}
```

Dostępność komponentów – dziedziczenie: tryp prywatny

```
class KlasaBazowa { // .....
        int _PolePryw_KBaz;
    protected :
        int _PoleChro_KBaz;
    public :
        int _PolePubl_KBaz;
}; // .....

class KlasaPochodna: private KlasaBazowa { } // .....

class KlasaPochodna_Nastepna: public KlasaPochodna { // .....
    public :
        void OdwolanieDoPol( );
}; // .....
```

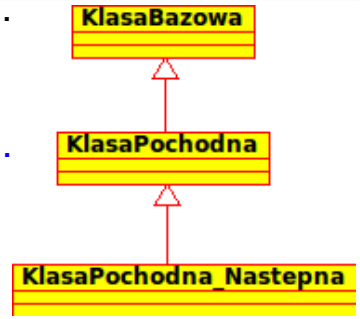



```
void KlasaPochodna_Nastepna::OdwolanieDoPol( )
{
    _PolePubl_KBaz = _PoleChro_KBaz = _PolePryw_KBaz = 0;
}
```

Dostępność komponentów – dziedziczenie: tryp prywatny

```
class KlasaBazowa { // .....
        int _PolePryw_KBaz;
    protected :
        int _PoleChro_KBaz;
    public :
        int _PolePubl_KBaz;
}; // .....

class KlasaPochodna: private KlasaBazowa { } // .....

class KlasaPochodna_Nastepna: public KlasaPochodna { // .....
    public :
        void OdwolanieDoPol( );
}; // .....

void KlasaPochodna_Nastepna::OdwolanieDoPol( )
{
    _PolePubl_KBaz = _PoleChro_KBaz = _PolePryw_KBaz = 0;
}
```

Jeżeli klasa bazowa jest dziedziczona w trybie prywatnym, to jej komponenty, które dostępne są w klasie pochodnej, nie będą dostępne w klasie dziedziczącej klasę pochodną (niezależnie do trybu dziedziczenia).



Koniec prezentacji
Dziękuję za uwagę