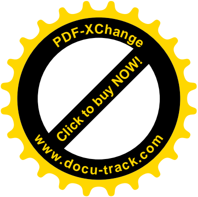
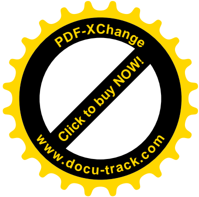


PDF-XChange Click to buy NOW! www.docu-track.com

Pogramowanie obiektów

1.1 Pojęcie klasy

Pojęcie klasy jest pojęciem zasadniczym w programowaniu obiektowym w C++. **Klasa** jest to zbiór obiektów. Na przykład klasa „Pojazd" jest zbiorem obiektów typu „Pojazd". **Instancją klasy** nazywa się obiekt przynależący do danej klasy. Mój pojazd jest więc instancją klasy „Pojazd". Dana klasa zawiera opis swoich instancji. Na przykład klasa „ Pojazd" zawiera opis swoich instancji, przez sam fakt, że jej nazwa brzmi „Pojazd" oraz ponieważ wiadomo, które obiekty są Pojazdami. Ten opis nazywa się **„abstrakcją"**, podobnie jak słowo *Pojazd* jest abstrakcją wszystkich pojazdów (**obiektem abstrakcyjnym**) w języku polskim.

W informatyce, pracuje się na „obiektach" informatycznych, które są reprezentacją obiektów(przedmiotów) świata rzeczywistego. Obiekty te są przechowywane w pamięci komputera. Przedstawione więc są one wyłącznie pod postacią informacji binarnej. W języku obiektowym, opis instancji nie ogranicza się jedynie do podania nazwy, jakimkolwiek przypadku opisywanym. Całkowity opis ten jest dokonywany za pomocą atrybutów i metod. Atrybuty przechowują wartości, a metody określają sposób przekształcania danych (sposób postępowania). **Opis obiektu** jest więc zadany przez dwa elementy języka programowania, tzn. Przez **dane** i przez **programy** działające na tych danych. W języku C++, opis instancji klasy jest dokonywany za pomocą atrybutów, tak jak w strukturach języka C oraz za pomocą metod, które są odpowiednikiem funkcji języka C. Definicja klasy w C++ sprowadza się więc do tego opisu i nadania nazwy. Tak więc, nie jest możliwe rozpoznanie zbioru instancji przynależących do klasy w C++. W rzeczywistości, potrzeba taka, jest rzadko potrzebna w programowaniu obiektowym.

1.2 Definicja klasy

Składnia określająca definicję klasy jest zbliżona do składni określającej strukturę w języku C. Definicja składniowa zawiera w sobie dwie części: nagłówek składający się ze słowa kluczowego **class**, po którym następuje nazwa klasy oraz ciało klasy ograniczone parą nawiasów klamrowych, po których następuje średnik.

```
class nazwa_klasy
  {
    //zbiór atrybutów i metod
  };
```

Przykładem definicji klasy może być klasa Punkt zdefiniowana w przykładzie 4.1

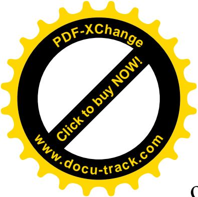
Przykład 4.1

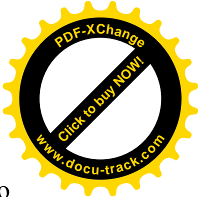
```
class PUNKT[1]
{
  int X,Y;
  void wstaw(int wx,int wy);
};
```

W klasie PUNKT zdefiniowano dwa atrybuty określające współrzędne, oraz metodę wstaw służącą do ustalenia współrzędnych punktu.

W przykładzie 4.1 metoda wstaw stanowi metodę abstrakcyjną, gdyż nie zawiera ciała metody (instrukcji wykonywanych przez tą metodę). Istnieją dwa sposoby na umieszczenie kody wykonywanego przez daną metodę. W przykładzie 4.2 klasa PUNKT zawiera dwie metody: wstaw i przesun. Ciało metody wstaw zawarte jest bezpośrednio w definicji klasy. Należy zwrócić uwagę, że w takim przypadku po nagłówku metody nie występuje średnik (porównaj z metodą przesun). Metoda przesun służy do przesunięcia wektora o pewien wektor o współrzędnych *[px,py]*. Nagłówek tej metody znajduje się wewnątrz definicji klasy, ponieważ nagłówki wszystkich metod należy umieszczać wewnątrz definicji klasy. Ciało metody przesun zostało zdefiniowane poza definicją klasy. W takim przypadku nazwa metody poprzedzona jest nazwą klasy i znakiem „::". Rozdzielenie to w pełni pasuje do wspomnianego wcześniej opisu klasy i programu wykonywanego na danych zawartych wdanej klasie. W wielu przypadkach opisane rozdzielenie opisu klasy i jego programu stosowane jest w taki sposób, że opis klasy umieszczany jest w **plikach nagłówkowych** (z rozszerzeniem *.hpp), natomiast definicja metod w plikach z rozszerzeniem *.cpp, o takiej samej nazwie co plik nagłówkowy. Taki komplet plików (.hpp i *.cpp) stanowi swoistą bibliotekę zawierającą zdefiniowane przez programistę

[1] W niniejszym opracowaniu autor założył, że nazwy klas będą pisane dużymi literami, natomiast obiekty (instancje klas) małymi.





obiekty, które można dowolnie wykorzystywać do tworzenia nowych programów (projektów), jak również do tworzenia nowych klas.

Przykład 4.2

```
class PUNKT
{
  int X,Y;
  void wstaw(int wx,int wy)
    {
     X=wx;
     Y=wy;
    };
  void przesun(int px,int py);
}; //koniec definicji klasy

//ciało metody przesun
void PUNKT::przesun(int px,int py)
  {
    X+=px;
    Y+=py;
  };
```

1.3 Tworzenie instancji

Instancja klasy może być stworzona domyślnie przez deklarację zmiennej w następujący sposób:
`nazwa_klasy nazwa_obiektu;`
gdzie `nazwa_klasy` zawiera nazwę klasy której instancją będzie obiekt o nazwie `nazwa_obiektu`.

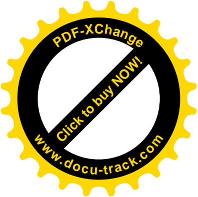
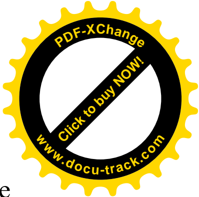
Przykład 4.3

```
#include <stdio.h>
...
//kod z przykładu 4.2
...
PUNKT punkt;

void main(void)
 {
    punkt.wstaw(10,20);
    punkt.przesun(10,10);
    printf(”Współrzędne punktu: %d,%d”,punkt.X,punkt.Y);
 };
```

W przykładzie 4.3 przedstawiono sposób wykorzystania obiektu. Rozpoczyna się deklaracją obiektu klasy `PUNKT` punkt, a następnie w funkcji main używając metod `wstaw` i `przesun` następuje zmiana współrzędnych danego obiektu. Należy zwrócić uwagę, że do metod jak również atrybutów odwołuje się za pomocą znaku „`.`”.

Przykład 4.4

```
#include <stdio.h>
...
//kod z przykładu 4.2
PUNKT *punkt;

void main(void)
 {
    punkt=new PUNKT;
    punkt->wstaw(10,20);
    punkt->przesun(10,10);
    printf(”Współrzędne punktu: %d,%d”,punkt->X,punkt->Y);
    delete punkt;
 };
```

Tak więc atrybuty i metody pozwalają modyfikować opis instancji. Jest również możliwe stworzenie instancji (zinstancjować, instancjowanie jako określenie czynności tworzenia) **dynamicznie**. W tym celu, należy zdeklarować wyłącznie wskaźnik na daną klasę, tak jak w przykładzie 4.4. Instancjowanie obiektu dynamicznego dokonuje się przy pomocy operatora **new** w następujący sposób:

```
nazwa_obiektu = new nazwa_klasy;
```

W instrukcji `punkt=new PUNKT` zmienna `punkt` jest referencją do instancji, która nie ma określonej nazwy. Dlatego, może ona zawierać referencję do jednej instancji, a później do innej instancji. Jeśli się dokona się następujących operacji:

```
punkt=new PUNKT;
punkt=new PUNKT;
```

W instrukcji `punkt=new PUNKT` zmienna punkt jest referencją do drugiej anonimowo stworzonej instancji. Do pierwszej instancji nie ma w takim przypadku odwołania(referencji). W języku C++, pamięć zajmowana przez tą pierwszą instancję jest bezpowrotnie tracona, ponieważ w przeciwieństwie do bardziej rozwiniętych języków obiektowych tj SMALTALK,BETA lub Eiffel, język C++ nie potrafi unieważnić automatycznie instancji bez referencji. Należy więc pamiętać aby **zawsze** unicestwiać instancje za pomocą operatora **delete**:

```
delete punkt;
```

Dostęp do atrybutów i metod w obiektach dynamicznych realizuje się za pomocą operatora „->".

1.4 Identyfikator domyślny: ***this***.

W definicji metody, ***this*** jest parametrem domyślnym, który zawiera referencję do obiektu, który otrzymuje odwołanie danej metody. Może on, dla przykładu, służyć do przekazywania referencji do tego obiektu innym metodom. Na przykład, jeżeli pragnie się wywołać funkcję, która przechowuje referencje w tablicy. Funkcja ta przechowuje obiekty klasy `PUNKT` i jest zagnieżdżona w klasie `WIELOKAT`:

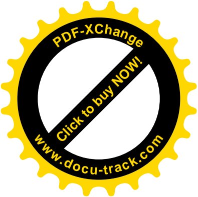
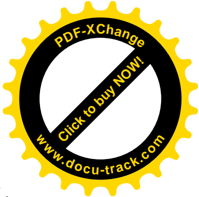
Przykład 4.5

```
class WIELOKAT
 {
    public:
        PUNKT punkty[10];
        short wskaznik;
        void WskPocz(void);
        void DodajPunkt(PUNKT nowypunkt);
 };

void WIELOKAT::WskPocz(void)
{
    wskaznik = 0;
};
void WIELOKAT:: DodajPunkt (PUNKT nowypunkt)
{
    punkty[wskaznik] = nowypunkt;
    wskaznik ++;
};
WIELOKAT trojkat; //zadeklarowany obiekt o nazwie trojkat
```

Jeśli stworzy się metodę `DodajDoListy` w klasie `PUNKT` :

Przykład 4.6

```
class PUNKT
  {
    ...
    void DodajDoListy(void);
  };

void PUNKT::DodajDoListy(void)
{
    trojkat.DodajPunkt (this);
}
```



PDF-XChange
Click to buy NOW!
www.docu-track.com

W metodzie DodajDoListy, występuje wywołanie metody DodajPunkt obiektu trojkat. Identyfikator **this** jest przekazywany jako parametr opisujący referencję do obiektu, który otrzymuje wywołanie metody DodajDoListy.

1.5 Hermetyzacja

Hermetyzacja jest bardzo ważną techniką programowania obiektowego. Hermetyzacja jest maskowaniem dostępu do pewnych atrybutów, metod instancji klasy. Hermetyzację w C++ realizuje się za pomocą słów kluczowych **public**, **protected** oraz **private**. Te słowa kluczowe pozwalają określić widoczność atrybutów i metod. Widoczność zdeklarowana przez słowo **public** jest widocznością globalną. Każda instancja będzie mogła mieć dostęp do atrybutów, których widoczność jest określona przez słowo kluczowe **public**. Widoczność określona przez słowo kluczowe **private** jest widocznością prywatną. Tylko instancje danej klasy będą miały dostęp do atrybutów, których widoczność określona jest słowo kluczowe **private**. Widoczność atrybutów w klasie domyślnie jest widocznością prywatną. Znaczenie słowa kluczowego **protected** jest równoważne znaczeniu słowa kluczowego **public** w danej klasie oraz znaczeniu słowa kluczowego **private** w podklasach danej klasy. Pojęcie podklasy będzie rozpatrywane w następnej części kursu.

Przykład 4.7

```
class PUNKT
{
   private:
     short X, Y;
   public:
     void przesun(short px, short py);
};
```

Atrybuty X i Y są atrybutami prywatnymi. Nie można mieć do nich dostępu za pomocą metod innej klasy.

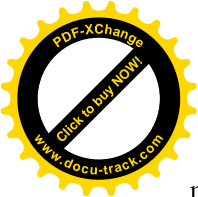
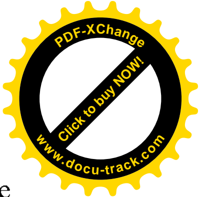
1.6 Konstruktory i destruktory

Konstruktor jest specyficzną funkcją danej klasy. Jego nazwa jest nazwą danej klasy. Jest on domyślnie wywoływany podczas definicji obiektu lub przydzielania pamięci pod obiekt przez operator **new**. Dla tej samej klasy może być zdefiniowanych kilka konstruktorów, pod warunkiem jednak, że typy parametrów konstruktorów pozwolą na ich rozróżnienie. Destruktor jest wywoływany podczas gdy zwalniana jeast pamięć przydzielana danemu obiektowi za pomocą operatora **delete**. Nazwa metody destructora poprzedzona jest znakiem „**~**”

Przykład 4.8

```
class PUNKT
{
   public:
     double  X, Y;
     PUNKT(double px  ;double py);
     PUNKT (PUNKT punkt);
    ~PUNKT(void){X=0; Y=0;};
};
PUNKT::PUNKT (double px  ;double py)
{
  X=px;
  Y=py;
}
PUNKT::PUNKT (PUNKT punkt)
{
  X=punkt.X;
  Y=punkt.Y;
}
```

W przykładzie 4.8, dla klasy PUNKT, jest możliwe nadanie początkowych wartości współrzednych podając jako parametr wywołania konstruktora współrzędne punktu (px py) albo inny obiekt klasy PUNKT.
Podczas **alokacji pamięci**(przydzielania pamięci) pod tworzoną instancję klasy PUNKT wywołanie odpowiedniego konstruktora zależy od tego, jaki zostanie użyty parametr wywołania. Występowanie wielu

metod o takich samych nazwach, różniących się ilością lub/i typem parametrów wywołania nosi nazwę **overloadingu.**

```
P1=new PUNKT (10,10);
P2=new PUNKT (P1);
```

Podczas wywoływania instrukcji **delete** P1 i **delete** P2 , a więc usuwania obiektów z pamięci (zwalniania pamięci zajmowanej przez obiekty), wywoływany jest destruktor, który dla przykładu 4.8 przypisuje współrzędnym punktów wartość 0.

1.7 Dziedziczenie

Powyżej opisano kilka właściwości i pojęć języka programowania C++ , a mianowicie: **klasy**, **atrybuty**, **metody**, **hermetyzację**, **konstruktory** i **destruktory**. W dalszej części opisana zostanie właściwość **dziedziczenia**, która to jest relacją pomiędzy **nadklasami** a **podklasami**, a także będziemy rozpratrywać wszystkie skutki i konsekwencje użycia tej relacji w połączeniu z pojęciami i właściwościami powyżej przytoczonymi.

1.7.1 Dziedziczenie proste

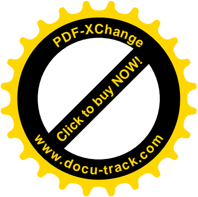
Dziedziczenie proste jest relacją pomiędzy dwoma klasami: podklasą i nadklasą. **Podklasa** reprezentuje pojęcie, pewną ideę w sposób bardziej wyspecjalizowany niż **nadklasa**, więc opis jej jest bogatszy. Dana podklasa dziedziczy atrybuty i metody swojej nadklasy. Jest możliwy dostęp do atrybutów i metod nadklasy instancji tej samej nazwy co atrybuty i metody zdefiniowane w samej klasie danej instancji.
Składnia dziedziczenia prostego jest następująca:

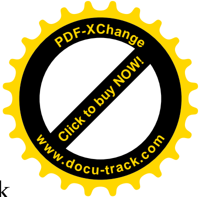
```
class nazwa_klasy : [public|private|protected] nazwa_drugiej_klasy
{
...
//ciało klasy
...
};
```

Może być wybrana jedna z opcji **public**, **private** lub **protected**, co nie jest jednak obowiązkowe. Domyślnie, gdy żadna z tych opcji nie jest wybrana, zastosowana będzie opcja **private**. W dalszej części tego kursu zostanie bardziej szczegółowo opisana składnia i opcjonalność tego wyrażenia. Jeśli zastosujemy powyższą składnię, to zaznaczymy w ten sposób, aby *klasa_pierwsza* dziedziczyła atrybuty i metody *klasy_drugiej*. W przykładzie 4.9 klasa WIELFOREMNY będzie dziedziczyła z klasy WIELOKAT. Jej opis składa się więc z atrybutów i metod zawartych w klasie WIELOKAT, a oprócz tego dołączone zostaną dwie nowe metody: obliczające pole i obwod tych figur oraz przedefiniowany konstruktor umożliwiający utworzenie różnych płaskich, foremnych figur geometrycznych tj. *trójkąt równoboczny, kwadrat, pięciokąt foremny* itd., na podstawie podanej liczby wierzchołków LW oraz długości boku DB.

Przykład 4.9

```
class WIELFOREMNY : public WIELOKAT
{
   protected:
    int bok;
    void UtworzFigure(void);
   public:
    WIELFOREMNY(int LW,int DB);
    double pole(void);
    double obwod(void);
 };

WIELFOREMNY::WIELFOREMNY(int LW,int DB)
 {
  wskaznik=LW;
  bok=DB;
  UtworzFigure();
 };
double WIELFOREMNY::obwod(void)
 {
   return wskaznik*bok;
 };
```





W konstruktorze parametr LW określający liczbę wierzchołków przypisywany jest atrybutowi wskaznik zdefiniowanemu w nadklasie WIELOKAT. Klasa WIELFOREMNY zawiera nowy atrybut o nazwie bok przechowujący długość boku. Został on zdefiniowany w sekcji chronionej (protected) w celu uniemożliwienia bezpośredniej zmiany jego wartości, gdyż musi ona pociągnąć za sobą zmianę położenia punktów. Metoda **double** obwod(**void**) oblicza długość obwodu danej figury[2]. Metoda **double** pole(**void**) oraz **void** UtworzFigure(**void**); zostały zdefiniowane jako metody **abstrakcyjne** (tym metodom przypisana tym metodom adres o wartości 0 a więc wywołanie takiej metody powoduje wystąpienie błędu lub sytuacji wyjątkowej), tzn. określające sposób ich użycia (liczbę parametrów, oraz zwracany typ) natomiast nie zawierają ciała (instrukcji przez nie wykonywanych).

1.7.2 Hermetyzacja i dziedziczenie

Najpierw należy powrócić do pojęć hermetyzacji atrybutów i metod, które były rozpatrywane w rozdziale 1.5. Możliwe jest postawienie przed definicją atrybutów i metod słowa kluczowego **public** lub **private**, po to, aby uczynić je dostępnymi lub niedostępnymi poza ciałami metod danej klasy. Tymczasem jednak, te dwa mechanizmy sterowania hermetyzacją, nie są wystarczające, przy uwzględnieniu w mechanizmach dziedziczenia.

Jeśli atrybut hermetyzacji jest ustawiony na **public**, jest on efektywnie widoczny w metodach podklas o tym samym nagłówku, który jest widoczny w całym programie. Jeśli atrybut hermetyzacji ustawiony jest private, nie jest on widoczny w metodach podklas o tym samym nagłówku, który nie jest widoczny w całym programie. Tymczasem byłaby bardziej interesująca sytuacja, w której można by maskować atrybut dla całości programu bez ukrywania go dla metod podklas. Ta ostatnia możliwość w C++ jest możliwa do zrealizowania za pomocą słowa **protected**. Jest ono używane w ten sam sposób, co słowa kluczowe **public** i **private**. (Przykład 4.9).

Przypadki dziedziczenia.

- Przypadek, w którym nadklasa jest dziedziczona w trybie prywatnym(słowo kluczowe **private** lub w ogóle nie określone).
  W tym przypadku wszystkie atrybuty dziedziczone stają się prywatne w danej podklasie.
- Przypadek, w którym nadklasa jest dziedziczona w trybie publicznym(słowo kluczowe **public**).
  W tym przypadku, wszystkie atrybuty dziedziczone zachowują swoje właściwości hermetyzacji w danej podklasie.
- Przypadek, w którym nadklasa jest dziedziczona w trybie chronionym(słowo kluczowe **protected**).
  W tym przypadku, atrybuty mające właściwość hermetyzacji publiczną (**public**) przyjmują w danej podklasie właściwość hermetyzacji chronioną (**protected**). Atrybut mający inną właściwość hermetyzacji, w danej podklasie ją zachowuje.

1.7.3 Funkcje wirtualne (*virtual*)

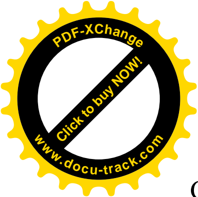
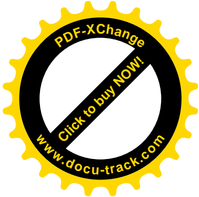
W podklasie nie można zdefiniować atrybutu lub metody o tej samej nazwie, co w dziedziczonej nadklasie. Można by z tego faktu wnioskować, że nie jest możliwe przedefiniowanie metody w C++. Tymczasem, jest możliwe przedefiniowanie metod wirtualnych, których definicja może ulec zmianie w podklasach. **Metoda wirtualna** jest zdefiniowana w klasie początkowej, a później przedefiniowywana w jej podklasach. Z punktu widzenia składni wyrażenia, metoda wirtualna jest definiowana tak, jak metoda klasyczna, przy czym prototyp definicji klasy jest poprzedzony przez słowo kluczowe **virtual**.

W podklasach, powtórna definicja danej metody powinna być przeprowadzana z tą samą liczbą parametrów o tym samym typie oraz z tym samym typem wartości zwracanej funkcji. Należy zauważyć, że słowo kluczowe **virtual** nie musi być koniecznie używane w danej podklasie, lecz użycie jego jest zalecane, dla czystości i przejrzystości kodu źródłowego programu.

W przykładzie 4.9 zdefiniowano podklasę WIELFOREMNY klasy WIELOKAT. Metoda pole służąca do obliczenia pola figury reprezentowanej przez dany obiekt klasy WIELFOREMNY nie została wyposażona w odpowiednie ciało gdyż prawdopodobnie w zależności od kształtu figury obliczenie pola będzie następowało w różny sposób. Wiedząc, że każda z figur tworzonych na bazie klasy WIELFOREMNY będzie wymagać definicji metody pole.Dlatego należy ją zdefiniować jako wirtualną, co wymaga utworzenia również jej ciała(nawiasy {} na końcu definicji):

```
virtual double pole(void){};
```

[2] Ponieważ w przypadku wielokątów foremnych wszystkie boki mają tę samą obwód liczony jest jako iloczyn długości boku i ilości boków występujących w danym przypadku wielokąta.

Całą klasę WIELFOREMNY należałoby zdefiniować tak jak w przykładzie 4.10:

Przykład 4.10

```
class WIELFOREMNY : public WIELOKAT
{
   protected:
    int bok;
    virtual void UtworzFigure(void);
   public:
    WIELFOREMNY(int LW,int DB);
    virtual double pole(void){};
    double obwod(void);
 };

WIELFOREMNY::WIELFOREMNY(int LW,int DB)
 {
  wskaznik=LW;
  bok=DB;
  UtworzFigure();
 };
double WIELFOREMNY::obwod(void)
 {
   return wskaznik*bok;
 };
void WIELFOREMNY::UtworzFigure(void)
 {
  //instrukcje tworzące daną figurę
  ...
 };
```

1.7.4 Dziedziczenie wielokrotne

W języku C++ istnieje pełna możliwość wykorzystywania dziedziczenia wielokrotnego. Umożliwia ona dziedziczenie atrybutów i metod z kilku nadklas. Składnia wyrażenia jest zbliżona do składni wyrażenia określającego przypadek dziedziczenia prostego. Każda nadklasa jest oddzielona przecinkiem od poprzedniej. Tak więc składnia wyrażenia jest następująca:

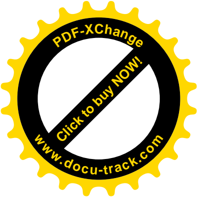
```
class nazwa_nowej_klasy:[public|private|protected]nadklasa1
                                  ,[public|private|protected] nadklasa2,…
```

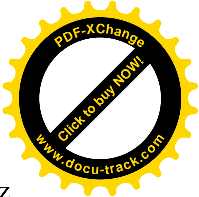
Dla każdej nadklasy, można zdefiniować tryb dziedziczenia, mianowicie: prywatny, publiczny lub chroniony. Odpowiedni tryb dziedziczenia spowoduje odpowiednie dziedziczenie atrybutów odziedziczonych danej klasy. Funkcjonowanie tego mechanizmu jest takie same jak w dziedziczeniu prostym. Można to zilustrować następującym przykładem.

W zdefiniowanej klasie WIELFOREMNY założono, że dana figura geometryczna będzie reprezentowana przez odpowiedni zbiór punktów stanowiących jej wierzchołki. Ponadto figury foremne posiadają środek(punkt równoodległy od każdego z wierzchołków) należałoby więc zdefiniować współrzędne środka figury. W tym celu można wykorzystać klasę PUNKT zdefiniowaną w przykładzie 4.2. Definicja tej klasy została zamieszczona w przykładzie 4.11.

Przykład 4.10

```
class WIELFOREMNY : public WIELOKAT,public PUNKT
{
   protected:
    int bok;
    virtual void UtworzFigure(void);
   public:
    WIELFOREMNY(int LW,int DB);
    virtual double pole(void){};
    double obwod(void);
};
```





W ten sposób klasa WIELFOREMNY posiada atrybuty i metody zawarte w klasach: WIELOKAT, oraz PUNKT.

1.8 Polimorfizm

Polimorfizm jest oparty na fakcie, że obiekt jest instancją swojej klasy (tej, której nazwa jest wykorzystywana podczas tworzenia obiektu za pomocą operatora **new**), lecz jest całkowicie instancją pośrednią nadklas tej bezpośredniej klasy, używanej do tworzenia instancji. Wartość pewnej zmiennej wskaźnikowej wskazująca na instancję jakieś podklasy może być przypisana wartości zmiennej wskaźnikowej wskazującej na instancję nadklasy. Ten sam wskaźnik może więc wskazywać na instancje, które nie mają tej samej klasy tworzenia tych instancji.

Pytania i zadania sprawdzające:

1. Co oznaczają następujące pojęcia: *klasa* i *obiekt* ?. Czym one się różnią?
2. Podaj definicję dowolnej podklasy klasy OBIEKT.
3. Jakie jest przeznaczenie indetyfikatora **this**?
4. Na czym polega maskowanie dostępu do atrybutów obiektu?
5. Omów główne założenia: hermetyzacji, dziedziczenia i polimorfizmu
6. Czy program napisany na podstawie przykładu 4.2 i 4.3 będzie działał? Uzasadnij odpowiedź.
7. Korzystając z klas zdefiniowanych w lekcji 4 utwórz klasę KWADRAT która będzie umożliwiać obliczanie obwodu, pola oraz długość przekątnej.
8. Utwórz klasę OKNO umożliwiającą dowolne ustalenie jego koloru, położenia jak i rozmiaru(szerokości i wysokości).
9. Utwórz klasę OKNOTEKSTOWE na bazie klasy z zadania 8. Klasa powinna mieć możliwość wyświetlania tekstu wewnątrz ramki(justowanego) w trybie tekstowym, oraz możliwość zmiany widoczności.
10. Utwórz klasę OKNOGRAFICZNE na bazie klasy z zadania 8. Klasa powinna mieć możliwość wyświetlania tekstu wewnątrz ramki(justowanego) w trybie graficznym, oraz możliwość zmiany widoczności.
11. Na podstawie klas: OKNO i OKNOGRAFICZNE z zadań 8 i 10 utwórz okno graficzne umożliwiające rysowanie figur geometrycznych tj. trójkąt, kwadrat, sześciokąt, okrąg.
12. Korzystając z klas zdefiniowanych w lekcji 4 utwórz klasę WEKTOR, która będzie umożliwiać podstawowe operacje i obliczenia na wektorach:
    a. Dodawanie wektorów
    b. Odejmowanie wektorów
    c. Iloczyn skalarny i wektorowy wektorów
    d. Kąt między wektorami
    e. Obrót wektora dookoła punktu o zadany kąt