Agregacja w klasach

Istotą programowania obiektowego jest wprowadzenie relacji pomiędzy klasami. Relacja może polegać np. na dodaniu elementów klasy *A* do klasy *B*. Dwie najważniejsze relacje to agregacja oraz dziedziczenie. Dziedziczenie zostawimy sobie na kolejne zajęcia, teraz zajmiemy się agregacją.

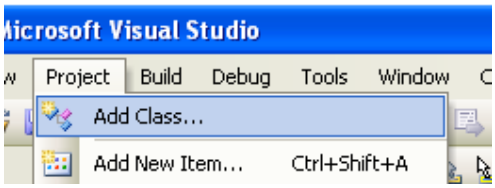
Agregacja to ogólnie łączenie klas w jedną klasę. Przykładem może być klasa *monitor*. Taka klasa opisuje urządzenie jakim jest monitor komputera. Monitor składa się z wielu elementów i pełni funkcję wyświetlania obrazu. Można stworzyć takie pola jak kolor monitora, kolor przycisków oraz metodę „wyświetl”. Oczywiście ktoś może dodać pole odnoszące się do właściwości samej matrycy, obudowy, zasilacza itp. Jednak wtedy nasza klasa osiągnie olbrzymie rozmiary, a co więcej, stracimy przejrzystość kodu. Lepiej stworzyć klasę opisującą matrycę czy obudowę, a następnie dodać pole typu *matryca* czy *obudowa* w deklaracji klasy *monitor* . Taka operacja to łączenie klas i nosi nazwę agregacji.

Przykład:

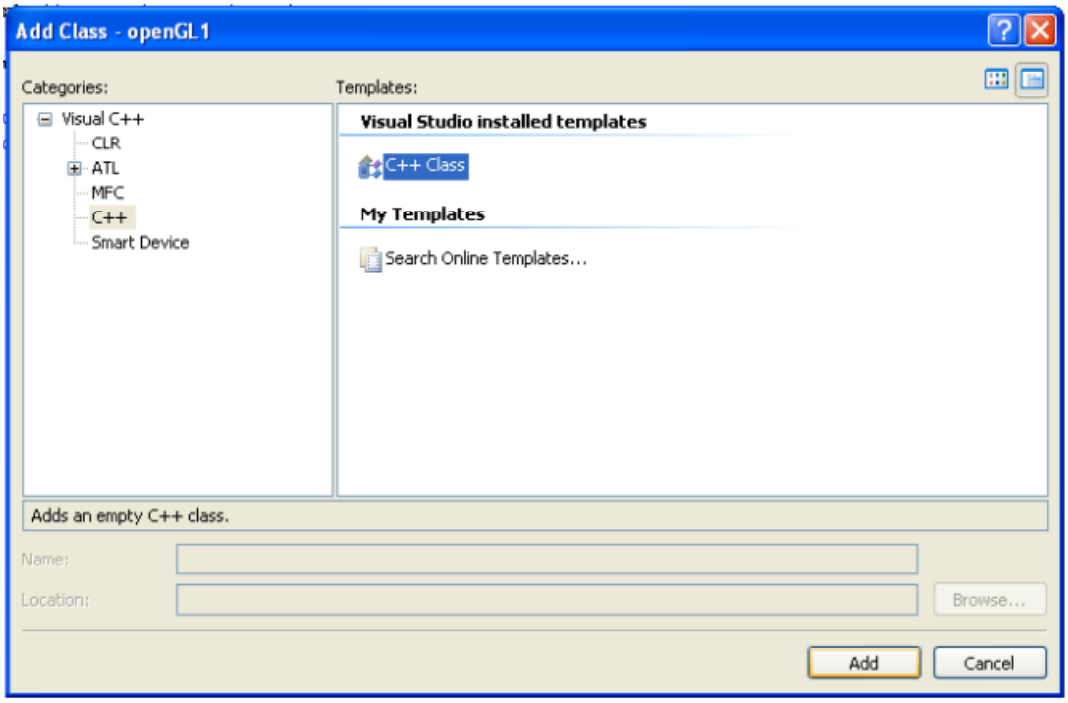
```
class matryca{
        public:
        int wielkość;
        char typ;
        int test(int czestotliwosc);
};

class monitor{
        public:
        int kolor;
        int cena;
        matryca  matryca_m;
        void wyswietl(void);
};
```

W dzisiejszym ćwiczeniu podobne zadanie tworzenia relacji wykonamy dla klasy *CPunkt* oraz *CKwadrat*. Klasa *CPunkt* posiada trzy pola: położenie *x, y ,z* oraz metody: *przesun(…)* i *rysuj(…).* Jest to logiczne, ponieważ punkt powinien mieć położenie zdefiniowane w układzie współrzędnych oraz możliwość jego przesunięcia jak i samego narysowania.

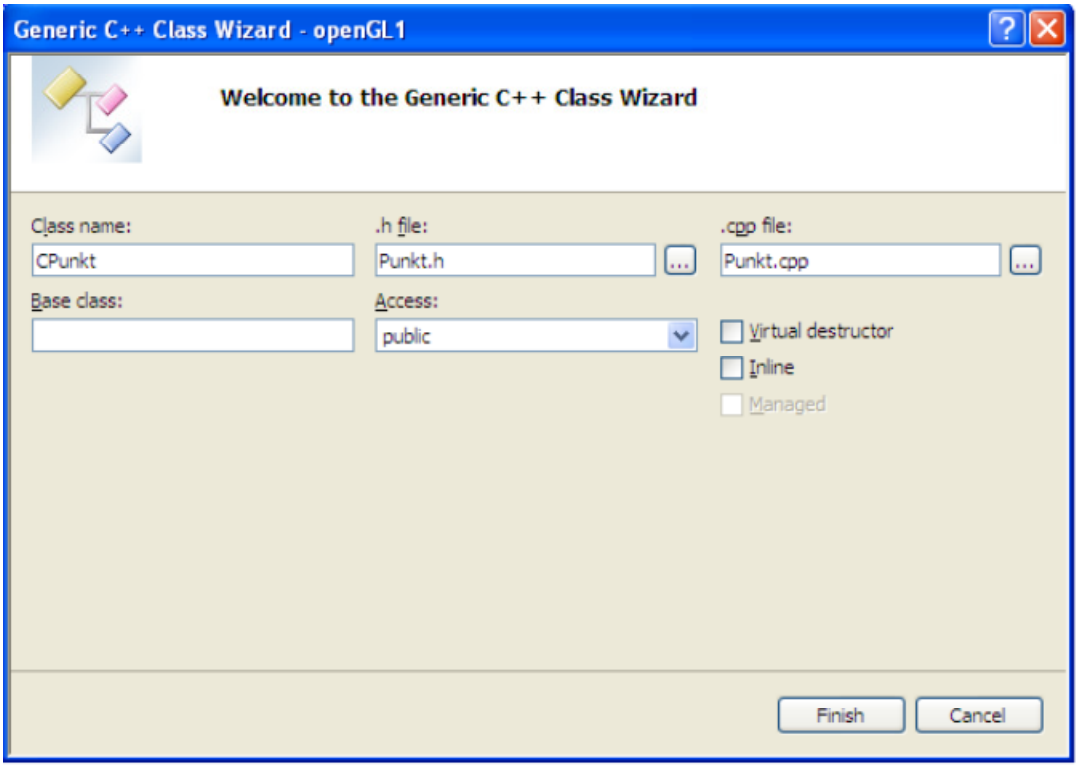
Teraz czas dołączyć klasę *Cpunkt* do naszego programu (z poprzednich zajęć). Uruchamiamy program a następnie dodajemy:



i wybieramy:



Oraz wpisujemy nazwę klasy:



Od tego momentu w naszym programie dostępne są dwa dodatkowe pliki tj. Punkt.h oraz Punkt.cpp. Standardowo w plikach tych jest odpowiednio deklaracja klasy oraz definicja konstruktora i destruktora klasy *CPunkt.* To oczywiście za mało dlatego teraz dodane zostaną kolejne metody i pola tej klasy. Pierwszą jednak rzeczą jest dołączenie biblioteki (w punkt.h):

```
#include <gl\glut.h> // glut.h includes gl.h and glu.h
```

A następnie zdeklarowanie naszego własnego konstruktora (przeładownie):

```
CPunkt(int x, int y, int z); //nasz konstruktor
```

Dodajmy też dwie nowe metody: rysuj(...) – rysuje punkt w przestrzeni, a jako parametry przyjmuje składowe r,g,b (kolor) punktu, metodę przesun, ktora przesuwa punkt w przestrzeni:

```
void rysuj(double r, double g, double b); // metoda rysujaca punkt
void przesun(int dx, int dy, int dz);//metoda zmieniajaca wspolrzedne punktu
```

Po uwzględnieniu zmian plik nagłowkowy punkt.h będzie wyglądał następująco:

```
#pragma once
#include <gl\glut.h> // glut.h includes gl.h and glu.h

class CPunkt
{
public: //metody i atrybuty dostepne dla wszystkich
	CPunkt(void); //konstruktor domyslny
	CPunkt(int x, int y, int z); //nasz konstruktor
public:
	~CPunkt(void);
	int pozycja[3]; //tablica ze wspolrzednymi punktu
	void rysuj(double r, double g, double b); // metoda rysujaca punkt
	void przesun(int dx, int dy, int dz);//metoda zmieniajaca wspolrzedne punktu

};
```

Teraz należy dodać definicje odpowiednich metod w pliku punkt.cpp.
Każda metoda napisana w pliku poprzedzona jest znakami ”Cpunkt::”, które definiują: “ta metoda dotyczy klasy CPunkt!”. Pisząc kod wewnątrz tak oznaczonej metody, aby odwołać się do atrybutów klasy CPunkt wystarczy napisać nazwę atrybutu np. aby użyć zmiennej *pozycja* wystarczy napisać jej nazwę (kompilator ma już informacje, że ta zmienna została zdefiniowana wewnątrz klasy CPunkt). Można również odwołać się poprzez: *this->pozycja.* Operator *this* mówi o tym, że odwołujemy się do konkretnego obiektu klasy.

Dodajmy więc definicję naszego konstruktora:

```
CPunkt::CPunkt(int x, int y, int z) // nasz konstruktor
{
// tutaj proszę dodać  kod, który będzie wpisywał do tablicy “pozycja” wartości początkowe x, y, z

}
```

oraz metodę rysującą punkt w przestrzeni:

```
void CPunkt::rysuj(double r, double g, double b){ //rysuje punkt w przestrzeni
	glColor3f(r,g,b); //ustalamy kolor punktu
	glBegin(GL_POINTS);//rysujemy punkt
	glVertex3f(pozycja[0], pozycja[1], pozycja[2]);
	glEnd();
}
```

i metodę przesuwającą punkt w przestrzeni:

```
void CPunkt::przesun(int dx, int dy, int dz){ //przesuwa punkt w przestrzeni
	pozycja[0]+=dx;
	pozycja[1]+=dy;
	pozycja[2]+=dz;
}
```

Teraz aby użyć klasy *CPunkt* w projekcie należy dołączyć odpowiedni plik nagłowkowy *(#include "Punkt.h"*) i stworzyć odpowiedni obiekt (*CPunkt a*; lub *CPunkt a(1,2,3);* gdzie 1,2,3 to współrzędne punktu w przestrzeni. W przypadku pierwszej metody tworzenia obiektu, początkowe współrzędne punktu są nieokreślone). Proszę teraz sprawdzić czy podczas kompilacji nie ma błędów, jeśli nie to przechodzimy do kolejnego kroku.

Naszym celem jest stworzenia relacji pomiędzy klasami. Skoro tak, to musimy stworzyć klasę, która będzie zawierała w sobie obiekt typu *CPunkt.* Taką klasą będzie dla nas klasa o nazwie *Ckwadrat*, opisująca figurę geometryczną – kwadrat.
Dodajmy do projektu nową klasę jak w poprzednim przykładzie, używając nazwy *CKwadrat*.
Do projektu zostanie dodany plik nagłówkowy *kwadrat.h*:

```
#pragma once
class CKwadrat
{
        public:
        CKwadrat(void); //domyslny konstruktor
        public:
        ~CKwadrat(void); //destruktor
};
```

Teraz należy się zastanowić jakie cechy opisują kwadrat. W tym przypadku mogą to być współrzędne wierzchołków (wierzcholek), kolor kwadratu (kolor) oraz zmienne określające jego przesunięcia w poszczególnych osiach (translacja i rotacja). Dodajmy też właściwość określającą czy kwadrat jest widoczny czy niewidzialny (hide).
Ponieważ stworzyliśmy klasę, która określa punkt w przestrzeni, użyjmy jej do zdefiniowania wierzchołków kwadratu. W tym celu dodajmy odpowiedni plik nagłówkowy:

```
#include "punkt.h"
```

Możemy teraz dodać do klasy tablice czteroelementową przechowującą współrzędne wierzchołków:

```
CPunkt wierzcholek[4]; // wspolrzedne wierzchole
```

Dodajmy też zmienne przechowujące informacje o kolorze, przesunięciu, rotacji i wyświetlaniu kwadratu. Niech dostęp do zmiennej określającej ukrycie/wyświetlenie figury będzie możliwy tylko z funkcji wewnątrz klasy – używamy słowa kluczowego "private":

```
double kolor[3]; // przechowuje informacje o skladowych red, green, blue koloru
double translacja[3]; // przechowuje wspolrzedne przesuniecia w trzech osiach
double rotacja[3]; //przechowuje wspolrzedne obrotow w trzech osiach
private:
bool hide; //okresla czy kwadrat ma byc wyswietlany 1 - nie, 0 tak
```

Zdefiniujmy nasz konstruktor:

```
CKwadrat(int x1, int y1, int x2, int y2,double r, double g, double b); // konstruktor, ktory automatycznie
przypisuje wartosci wierzcholkom i ustala
kolor kwadratu
```

Niech jako parametry przyjmuje współrzędne lewego dolnego i prawego górnego rogu kwadratu (to wystarczy aby określić współrzędne pozostałych wierzchołków), oraz składowe rgb koloru w jakim będzie wyświetlany kwadrat. Dodajmy trzy funkcje: rysującą, ukrywającą i pokazującą kwadrat:

```
void Rysuj(); // funkcja rysuj1ca figure
void Ukryj(); // wylaczenie wyswietlania figury
void Pokaz(); // wylaczenie wyswietlania figury
```

Przydatne też będą metody przesuwające i obracające dana figurę:

```
void Przesun(int dx, int dy, int dz); //przesuniecie figury
void Obroc(double alfa, double beta, double gamma); // obrot figury
```

Ostatecznie plik nagłówkowy klasy *CKwadrat* wygląda następująco:

```
#pragma once
#include "punkt.h"
class CKwadrat
{
public:
	CKwadrat(void); //domyslny konstruktor
	CKwadrat(int x1, int y1, int x2, int y2,double r, double g, double b); // konstruktor, ktory
automatycznie przypisuje wartosci wierzcholkomi ustala kolor kwadratu
public:
	~CKwadrat(void); //destruktor
	void Rysuj(); // funkcja rysuj1ca figure
	void Ukryj(); // wylaczenie wyswietlania figury
	void Pokaz(); // wylaczenie wyswietlania figury
	void Przesun(int dx, int dy, int dz); //przesuniecie figury
	void Obroc(double alfa, double beta, double gamma); // obrot figury
	CPunkt wierzcholek[4]; // wspolrzedne wierzcholkow
	double kolor[3]; // przechowuje informacje o skladowych red, green, blue koloru
	private:
	bool hide; //okresla czy kwadrat ma byc wyswietlany 1 - nie, 0 tak
	public:
	double translacja[3]; // przechowuje wspolrzedne przesuniecia w trzech osiach
	double rotacja[3]; //przechowuje wspolrzedne obrotow w trzech osiach
};
```

Teraz czas napisać definicję dla powyższych deklaracji:

```
CKwadrat::CKwadrat(int x1, int y1, int x2, int y2,double r, double g, double b){ // tutaj proszę dodać konstruktor,
ktory automatycznie przypisuje wartosci wierzcholkom i ustala kolor kwadratu na podstawie wartości
przekazanych do funkcji, oraz zeruje wektor przesuniecia i rotacji. Powinien rownież
ustawiać zmienną hide na 0.
}
```

Napiszmy metodę która rysuje nasz kwadrat:

```
void CKwadrat::Rysuj(){ // funkcja rysuj1ca figure
	glPushMatrix();// save matrix prior to transform, pozwala na zmiane pozycji i orientacji tylko jednego
obiektu
	glTranslatef(translacja[0],translacja[1],translacja[2]); //translacja I rotacje:
	glRotatef( rotacja[0], 1.0, 0.0, 0.0 );//rotate about the X axis
	glRotatef( rotacja[1], 0.0, 1.0, 0.0 );//rotate about the Y axis
	glRotatef( rotacja[2], 0.0, 0.0, 1.0 );//rotate about the Z axis
	if (!hide) {
		glColor3f(kolor[0],kolor[1],kolor[2]);//ustawienie koloru
		glBegin( GL_QUADS ); //rysujemy czworokąt
		glVertex3f( wierzcholek[0].pozycja[0], wierzcholek[0].pozycja[1],wierzcholek[0].pozycja[2] );
		glVertex3f( wierzcholek[1].pozycja[0], wierzcholek[1].pozycja[1],wierzcholek[1].pozycja[2] );
		glVertex3f( wierzcholek[2].pozycja[0], wierzcholek[2].pozycja[1],wierzcholek[2].pozycja[2] );
		glVertex3f( wierzcholek[3].pozycja[0], wierzcholek[3].pozycja[1],wierzcholek[3].pozycja[2] );
		glEnd();
	}
	glPopMatrix();// restore matrix after transform, koniec przekształcen na wybranym obiekcie
}
```

Następnie dodajmu do klasy CKwadrat metody ukrywające i pozwalajace na ponowne wyświetlenie kwadratu:

```
void CKwadrat::Ukryj(){ // wylaczenie wyswietlania figury
hide=1;
```

```
}
void CKwadrat::Pokaz(){ // wylaczenie wyswietlania figury
hide=0;
}
```

Na końcu pozostały metody które zmieniają odpowiednio wektor translacji i rotacji figury:

```
void CKwadrat::Przesun(int dx, int dy, int dz){ //przesuniecie figury
	translacja[0]+=dx;
	translacja[1]+=dy;
	translacja[2]+=dz;
}
void CKwadrat::Obroc(double alfa, double beta, double gamma){ // obrot figury
	rotacja[0]+=alfa;
	rotacja[1]+=beta;
	rotacja[2]+=gamma;
}
```

Gotowe! Dodaliśmy do projektu dwie klasy. Teraz należy z nich prawidłowo skorzystać. Chcemy umieścić dwa kwadraty na naszej scenie openGL. Ponieważ klasa *openGLview* reprezentuje naszą scenę, dodajmy do niej dwie zmienne klasy *CKwadrat*. W pliku nagłówkowym klasy (*openGLview.h*) dodajemy:

```
public:
CKwadrat * kwadrat; // kwadrat nr 1
CKwadrat * kwadrat2; // kwadrat nr 2
```

Niech to będą zmienne typu wskaźnikowego o dostępie dla wszystkich (public).
Teraz musimy stworzyć oba obiekty. Niech będą one tworzone w momencie powstawania obiektu klasy *openGLview* czyli w jej konstruktorze:

```
openGLview::openGLview(void)
{
kwadrat = new CKwadrat(-20,-20,20,20,1,1,0);
kwadrat2 = new CKwadrat(-25,-25,25,25,0,1,0);
}
```

Jak widać użyliśmy naszego wcześniej zdefiniowanego konstruktora nadając obu kwadratom pewne wymiary i kolor. Jak widać przydaje się przeciążanie konstruktorów – w jednej linijce mamy zdefiniowany odpowiedni obiekt.
Ponieważ nasz obiekt jest tworzony dynamicznie, musimy pamiętać o jego usunięciu.
Najwygodniej jest skorzystać z destruktora klasy w której używamy obiektu klasy *CKwadrat*:

```
openGLview::~openGLview(void)
{
delete kwadrat;
delete kwadrat2;
}
```

Teraz pozostaje nam wywoływanie metod, które rysują kwadraty. Użyjmy stworzonej wcześniej funkcji *rysuj_figury().* Wstawmy do niej wywołanie odpowiednich metod:

```
kwadrat->Rysuj(); //rysuje obiekt kwadrat
kwadrat2->Rysuj(); //rysuje obiekt kwadrat 2
```

Pozostałe wywołania z funkcji *rysuj_figury()* można usunąć.
Teraz można skompilować i… otrzymać błąd. Oczywiście musimy jeszcze dołączyć plik nagłówkowy `#include "Kwadrat.h"` w pliku nagłówkowym klasy *openGLview.*
Gotowe! Na naszej scenie rysują się dwa kwadraty. Zysk jaki osiągneliśmy to przejrzystość kodu.
Do wyświetlenia całej figury używamy tylko jednej linijki kodu. Dzięki klasom, nawet gdyby figura była bardzo skomplikowana, nadal będzie to jedna linijka. Cały kod, który rysuje figurę umieszczony jest wewnątrz klasy.

Tworząc klasę *CKwadrat* korzystaliśmy z klasy *Cpunkt*, następnie tworząc klasę *openGLview* używaliśmy obiektów typu *CKwadrat*. Zjawisko to nosi nazwę agregacji. W ten sposób budując skomplikowane obiekty rozdzielamy je na mniejsze i z nich budujemy bardziej złożone. np. Tworząc klasę *Ckwadrat* można korzystać z klasy *Clinia,* a przy jej tworzeniu skorzystać z klasy *Cpunkt*. Taki sposób postępowania pozwala zachować przejrzystość kodu w dużych projektach i nie tylko!

Już na zakończenie spróbujemy obsłużyć klawiaturę w openGL:
Do klasy *openGLview* dodajmy deklarację metody obsługującej klawiaturę:

```
void keyboard(unsigned char key, int x, int y);
```

oraz jej definicję w pliku *openGLview.cpp*:

```
void openGLview::keyboard(unsigned char key, int x, int y)// metoda w zależności od wciśniętego przycisku wywołuje odpowiednie zdarzenia
{
	if (key == 27) { exit(0); // escape key is ascii 27 }
	if (key == 97) { //key a kwadrat->Przesun(10,0,0); }
	if (key == 115) { //key s kwadrat->Przesun(0,10,0); }
	if (key == 100) { //key d kwadrat->Przesun(0,0,10); }
	if (key == 122) { //key z kwadrat->Przesun(-10,0,0); }
	if (key == 120) { //key x kwadrat->Przesun(0,-10,0); }
	if (key == 99) { //key c kwadrat->Przesun(0,0,-10); }
	if (key == 102) { //key f kwadrat->Obroc(10,0,0); }
	if (key == 103) { //key g kwadrat->Obroc(0,10,0); }
	if (key == 104) { //key h kwadrat->Obroc(0,0,10); }
	if (key == 118) { //key v kwadrat->Obroc(-10,0,0); }
	if (key == 98) { //key b kwadrat->Obroc(0,-10,0); }
	if (key == 110) { //key n kwadrat->Obroc(0,0,-10); }
	if (key == 111) { //key o kwadrat->Ukryj(); }
	if (key == 112) { //key o kwadrat->Pokaz(); }
}
```

Aby wskazać funkcję obsługującą klawiaturę dodajmy linię w klasie związanej z naszym formularzem (openGL1Dlg):

```
void keyboard(unsigned char key, int x, int y)
{
	gl.keyboard(key, x, y);
}
```

Oczywiście, tak jak w przypadku obsługi wyświetlania oraz zmiany wymiarów, należy obsłużyć funkcję klawiatury dopisując:

```
glutKeyboardFunc(keyboard);
```

W definicji funkcji `void CopenGL1Dlg::OnBnClickedOk()`

Sprawozdanie:

1. Ile razy i gdzie dokonywana jest agregacja klas w naszym programie?
2. Podaj definicję konstruktora przeładowanego: *CKwadrat::CKwadrat(int x1, int y1, int x2, int y2,double r, double g, double b)*
3. Podaj przykład agregacji na dowolnych dwóch klasach (o kilku metodach i polach), uzasadnij użycie agregacji
4. Czy można użyć konstruktor klasy podrzędnej do inicjalizacji właściwości obiektu z klasy podrzędnej w klasie w której agregujemy? (przykład: klasa podrzędna to *CPunkt*, a klasa w której agregujemy to *CKwadrat*)?
5. Jak można odnieść się do pola klasy *CPunkt* w metodzie klasy *CKwadrat*? Podaj przykład zrealizowany podczas zajęć. Sprawdź czy ta sama zasada obowiązuje dla kolejnej klasy w hierarchii agregacji (np. *CInfo<>CPunkt<>CKwadrat*).