Laboratorium Informatyki II
Ćwiczenie 10

Dziedziczenie

Przyjaźń pomiędzy klasami daje dostęp do cech i metod prywatnych ale wymaga utworzenia w klasie zaprzyjaźnionej obiektu klasy nadającej status przyjaźni. Jest to w wielu przypadkach nieefektywne i niewygodne w implementacji. Optymalnym rozwiązaniem byłoby przeniesienie cech i metod z klasy do klasy. Efekt ten bez konieczności fizycznego przepisywania kodu zapewnia mechanizm **dziedziczenia**.

W procesie dziedziczenia klasa **rodzica** przekazuje klasie **potomnej** swoje cechy i metody (z wyłączeniem *konstruktorów* i *destruktora*). Klasa może dziedziczyć po więcej jak jednej klasie. Mechanizm ten zapewnia efektywną metodę ponownego wykorzystania kodu.

Jeżeli w klasie potomka zostanie zdefiniowania cecha lub metoda o tej samej nazwie co w klasie rodzica to będą one przesłaniały. Dostęp do elementów dziedziczonych z klasy rodzica będzie możliwy za pomocą operatora zasięgu "`::`".

W procesie dziedziczenia istotne staje się określenie w jaki sposób odziedziczone elementy klasy mają być dostępne przy odwoływaniu się do elementów obiektu klasy potomnej. Przy dziedziczeniu wykorzystuje się trzeci poziom dostępu przy definiowaniu klasy, obszar chroniony `protected:`. Elementy z tego poziomu dostępu w obiekcie utworzonym na bazie klasy dostępne są tak jak gdyby były zdefiniowane na poziomie `private:`, natomiast w kodzie klasy potomnej są one dostępne tak jakby były w trybie `protected:`. W zależności od trybu dziedziczenia dostęp do elementów klasy bazowej (rodzica) zmienia się zgodnie z opisem w poniższej tabeli.

		Sposób dziedziczenia		
		public	**protected**	**private**
W klasie bazowej	**public**	public	protected	private
	protected	protected	protected	private
	private	niedostępne	niedostępne	niedostępne

Wybór trybu dziedziczenia należny dobrać optymalnie do implementowanego algorytmu decydując które elementy klasy rodzica mają być dostępne w kodzie klasy pochodnej a które jako elementy jej obiektu. Klasa może dziedziczyć po więcej jak jednej klasie. W przykładowym kodzie pokazanym poniżej zastosowano dziedziczenie.

```
#include <iostream>
using namespace std;
class punkt
{
public:
    punkt(int x, int y)
    {
        this->x=x;
```

```
        this->y=y;
    }
    void zaczep()
    {
        cout << "Punkt --> (" << x << "," << y << ")" << endl;
    }
protected:
    int x,y;
};
class wektor
{
protected:
    int dx,dy;
public:
    wektor(int dx, int dy)
    {
        this->dx=dx;
        this->dy=dy;
    }
    int get_dx(){
        return this->dx;
    }
    int get_dy(){
        return this->dy;
    }
};
class prostokat :  private punkt, public wektor
{
public:
    prostokat(int x, int y, int dx, int dy) : punkt(x,y),wektor(dx,dy)
    {
        this->x=x;
        this->y=y;
        this->dx=dx;
        this->dy=dy;
    }
    prostokat operator<<(wektor A)
    {
        prostokat B(x,y,dx,dy);
        B.x=x+A.get_dx();
        B.y=y+A.get_dy();
        return B;
    }
    void showProstokat()
    {
        cout << "Prostokąt --> (" << x << ","
             << y << ":" << dx << "," << dy << ")" << endl;
    }
};
```

```
int main()
{
    prostokat A(0,0,10,20);
    A.showProstokat();
    wektor B(12,32);
    A = A<<B;
    A.showProstokat();
}
```

W powyższym przykładzie klasa `prostokat` dziedziczy w trybie prywatnym po klasie `punkt` i trybie publicznym po klasie `wektor`. Należy zwrócić uwagę na **konstruktor** klasy prostokąt w którym realizowane jest odwołanie do konstruktorów klasy **punkt** i **wektor**.

Zadania

W oparciu o informacje wykładowe i treść niniejszej instrukcji napisać program obiektowy przetwarzający macierze. W programie zaimplementować dwie klasy:

1. klasa podstawowa definiująca:
   - macierz prostokątną o określonej liczbie wierszy i kolumn
   - konstruktor definiujący obiekt o podanej liczbie wierszy i kolumn wypełniony zerami
   - metody publiczne: ustawiającą elementy macierzy, wyświetlającą macierz
2. klasę pochodną definiującą macierz kwadratową zawierającą dodatkowo:
   - konstruktory lub metody ustawiające macierz jako jedynkową i diagonalną
3. klasę pochodną umożliwiającą wektor
4. zaimplementować w wybranych optymalnie klasach działania na macierzach:
   - mnożenia, dodawania (wybrane warianty)
   - transpozycji macierzy
   - opcjonalnie także: wyznacznik macierzy i macierz odwrotną