Laboratorium Informatyki II
Ćwiczenie 11

1 Polimorfizm

Polimorfizm czyli wielopostaciowość, jest zborem metod umożliwiających zapisanie funkcji (metod) pod wieloma postaciami, dzięki czemu można za ich pomocą przetwarzać wiele rożnych typów danych bez konieczności znajomości tych typów. Polimorfizm dzieli się na statyczny który obejmuje przeciążanie funkcji, metod i operatorów oraz polimorfizm dynamiczny dotyczący metod wirtualnych.

1.1 Polimorfizm dynamiczny

Polimorfizm dynamiczny opiera się na dziedziczeniu i metodach wirtualnych. Zapewnia wygodną metodę skalowania kodu, rozszerzania funkcjonalności tworzonego programu o nowe elementy realizowane przez klasy dziedziczące po klasie **abstrakcyjnej**.

Procedura opiera się na klasie abstrakcyjnej po której dziedziczą pozostałe klasy zdefiniowane w programie. Na bazie tej klasy nie tworzy się obiektów, jest ona stosowana do zautomatyzowanego odwołania się do obiektów utworzonych na bazie klas pochodnych. Realizowane jest to przez metody wirtualne o tej samej nazwie i składni wywołania.

W klasie abstrakcyjnej deklaruje się metodę lub metody wirtualne puste lub z kodem "domyślnym", realizowanym gdy w klasie pochodnej nie zadeklarowano odpowiedniej metody wirtualnej.

```
#include <iostream>
class bazowa
{
protected:
    int x;
public:
    bazowa(int x)
    {
        this->x = x;
    }
    virtual float suma(int a)
    {
        float y=0;
        for(int i=0;i<a;i++)
            y+=x*i;
        return y/a;
    }
};
class dane_A : public bazowa
```

```
{
private:
    int x;
public:
    dane_A(int x):bazowa(x)
    {
        this->x=x;
    }

};
class dane_B : public bazowa
{
private:
    float *x;
    int n;
public:
    dane_B(int n) : bazowa(n)
    {
        this->n=n;
        x = new float[n];
        for(int i=0;i<n;i++)
            x[i] = i+n;
    }
    virtual float suma(int a)
    {
        float y=0;
        for(int i=0;n>a?i<a:i<n;i++)
            y+=x[i];
        return y/a;
    }
};
int main()
{
    bazowa *A = new dane_A(12);
    bazowa *B = new dane_A(5);
    bazowa *C = new dane_B(3);
    bazowa *D = new dane_B(10);
    bazowa *zestaw[]{A,B,C,D};
    for(int i=0;i<4;i++)
    {
        std::cout << "Suma: " << zestaw[i]->suma(5) << std::endl;
        std::cout <<"------------------------" << std::endl;
    }
}
```

Alternatywny kod funkcji `main()`.

```
int main()
{
    bazowa **zestaw = new bazowa*[4];
    zestaw[0] = new dane_A(12);
```

```
    zestaw[1] = new dane_A(5);
    zestaw[2] = new dane_B(3);
    zestaw[3] = new dane_B(10);
    for(int i=0;i<4;i++)
    {
        std::cout << "Suma: " << zestaw[i]->suma(5) << std::endl;
        std::cout <<"-----------------------" << std::endl;
    }
}
```

W powyższym przykładzie w klasie abstrakcyjnej zdefiniowano metodę wirtualną o nazwie `suma` zwracającą wartość typu `float` i przyjmującą jeden parametr typu `int`. W klasach pochodnych zdefiniowano metody o tej samej nazwie, zwracające wartość tego samego typu i przyjmująca parametr wejściowy tego samego typu. W przykładzie w jednej z klas nie zdefiniowano metody wirtualnej. Dla obiektu tej klasy wykonana zostanie metoda z klasy abstrakcyjnej.

W programie dynamicznie utworzono cztery obiekty klasy abstrakcyjnej w oparciu o klasy pochodne. W tak zaprojektowanym programie można łatwo dodać nowe klasy pochodne i obiekty na ich bazie utworzone i włączyć jako kolejne elementy projektu.

Metodę wirtualną w klasie abstrakcyjnej można zdefiniować jako "pustą", jak pokazano w kodzie poniżej, ale w takim przypadku klasy pochodne muszą posiadać definicję metody wirtualnej.

```
class bazowa
{
public:
    virtual float suma(int a)=0
};
```

2 Szablon

Zdefiniowane funkcje i klasy operują na zadeklarowanych typach danych. W wielu sytuacjach gdy zachodzi konieczność zastosowania innych typów danych przeciąża się funkcje lub definiuje nową klasę w oparciu o wymagany typ danych.

Rozwiązaniem tego problemu jest zaprogramowanie adaptacyjnych funkcji i klas w oparciu o szablon z adaptacyjnymi typami danych. Definiuje się go za pomocą słowa kluczowego `template` i dwóch metod definiowania typów adaptacyjnych `typename` lub `class`, oba są sobie równoważne. Deklarację szablonu umieszcza się bezpośrednio przed wierszem z deklaracją funkcji lub klasy, jak pokazano poniżej:

```
template<typename zm1, typename zm2>
lub
template<class zm1,class zm2>
```

2.1 Funkcji

Szablon funkcji można zdefiniować na kilka sposobów w zależności od liczby zdefiniowanych w szablonie typów:

```
typ_A funkcja(); // funkcja zwraca wartość typu adaptacyjnego
typ_A funkcja(typ_A x); // funkcja zwraca i przyjmuje wartość typu
```

```
                              adaptacyjnego
typ_A funkcja(typ_B x); // funkcja zwraca wartość jednego a przyjmuje wartość
                              innego (innych) typów adaptacyjnych.
typ_A funkcja(typ_A x, typ_B y); // funkcja zwraca wartość jednego a przyjmuje
                                    wartość tego samego i innego (innych)
                                    typów adaptacyjnych.
int funckja(typ_A x); // funkcja zwraca wartość jednego z typów standardowych
                         (lub jest typu void) a przyjmuje atrybut(ty) typu
                         adaptacyjnego
```

Wywołanie funkcji szablonowej może zostać zrealizowane na dwa sposoby w zależności od tego czy adaptacyjny typ jest zarówno atrybutem jak i wielkością zwracaną. W takim wypadku funkcja potrafi przypisać do typu szablonowego właściwy typ na podstawie typu atrybutu. W innym przypadku należy w nawiasach ostrych podać listę typów przypisywanych w danym wywołaniu do typów adaptacyjnych.Elementem szablonu może być zmienna dowolnego typu nie wykorzystywana w deklaracji funkcji, a w jej kodzie. Wartość tego parametru jest przekazywana jako atrybut inicjujący szablon funkcji, jak pokazano w poniższym przykładzie kodu.

```
#include <iostream>
using namespace std;
template<typename typ_A>
typ_A suma(typ_A x,int n)
{
    typ_A suma = x;
    for(int i=1;i<n;i++)
        suma+=x;
    return suma;
}
template<typename typ_A, int n>
typ_A suma(typ_A x)
{
    typ_A suma = x;
    for(int i=1;i<n;i++)
        suma+=x;
    return suma;
}
int main()
{
    string v = "Suma";
    float x = 3.1415;
    int y = 508;
    int n = 6;
    cout << "Suma " << n << " elementów \"" << v << "\" wynosi: "
         << suma(v,n) << endl;
    cout << "Suma " << n << " elementów \"" << x << "\" wynosi: "
         << suma(x,n) << endl;
    cout << "Suma " << n << " elementów \"" << x << "\" wynosi: "
         << suma<int>(x,n) << endl;
    cout << "Suma " << n << " elementów \"" << y << "\" wynosi: "
         << suma(y,n) << endl;
```

```
    cout << "Suma " << n << " elementów \"" << x << "\" wynosi: "
         << suma<float,6>(x) << endl;
}
```

2.2 Klasy

W przypadku szablonu klasy konieczne jest określenie typów adaptacyjnych w momencie deklaracji obiektu. Nie jest możliwe automatyczne wykrywanie typów za pomocą atrybutów inicjujących obiekt.

```
#include <iostream>
using namespace std;
template<typename typ_A>
class liczby
{
private:
    typ_A *tab;
    int n;
public:
    ~liczby()
    {
        delete [] tab;
    }
    liczby(int n)
    {
        this->n = n;
        tab = new typ_A[n];
        for(int i=0;i<n;i++){
            cout << "podaj [" << i+1 << "] wartość: ";
            cin >> tab[i];
        }
    }
    void drukuj()
    {
        for(int i=0;i<n;i++){
            cout << i+1 << " --> " << tab[i] << endl;
        }
    }
};
int main()
{
    liczby<string> A(3);
    liczby<float> B(5);
    A.drukuj();
    B.drukuj();
}
```

Podobnie jak w przypadku szablonu funkcji także w przypadku klasy za pośrednictwem szablonu można przekazać do klasy parametr.

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

```
template<typename typ_A, int n>
class liczby
{
private:
    typ_A *tab;
public:
    ~liczby()
    {
        delete [] tab;
    }
    liczby()
    {
        tab = new typ_A[n];
        for(int i=0;i<n;i++){
            cout << "podaj [" << i+1 << "] wartość: ";
            cin >> tab[i];
        }
    }
    void drukuj()
    {
        for(int i=0;i<n;i++){
            cout << i+1 << " --> " << tab[i] << endl;
        }
    }
};
int main()
{
    liczby<string,3> A;
    liczby<float,5> B;
    A.drukuj();
    B.drukuj();
}
```

Zadania

W oparciu o informacje wykładowe i treść niniejszej instrukcji napisać:

A) program z wykorzystaniem polimorfizmu dynamicznego do katalogowania i przeglądania asortymentu wybranego sklepu

B) projekt szablonu klasy pozwalającej na przechowywanie i przetwarzanie grup asortymentu z zadania A)