Polimorfizm

Polimorfizm

Polimorfizm (wielopostaciowość) jest jednym z filarów obiektowości (obok abstrakcji, dziedziczenia i hermetyzacji). Dzięki temu mechanizmowi możliwa jest zmiana zachowania tych samych wirtualnych funkcji składowych w czasie wykonywania programu.

Aby polimorfizm działał, potrzebne nam są dwie najważniejsze rzeczy:

• wskaźnik lub referencja do obiektu typu bazowego

• określenie wybranej funkcji jako *virtual* (wirtualna)

Nie zapominajmy, że bez dziedziczenia polimorfizm również nie miałby sensu! :)

Przykład 1

Kod

```
#include <iostream>
#define PI 3.14

// KLASA BAZOWA
class Shape
{
protected:
    double height;
    double width;
public:
    Shape(double length): height(length), width(length) {}
    Shape(double h, double w): height(h), width(w) {}
    virtual double area()   // funkcja wirtualna
    {
        return height * width;
    }
};

// KLASA POCHODNA
class Circle : public Shape
{
public:
    Circle(double w): Shape(w) {}
    double area()
    {
        return PI * (width /2 ) * (width / 2);
    }
};

// ZWYKŁA FUNKCJA
void ShowArea(Shape& shape)  // referencja do obiektu typu bazowego
{
    std::cout << "Area : " << shape.area() << std::endl;  // wywołanie funkcji wirtualnej
}

int main()
{
    Shape rectangle(10, 5);
    Circle circle(10);

    ShowArea(rectangle);
    ShowArea(circle);

    return 0;
}
```

Widok konsoli

Wyjaśnienie

Jak widzimy - wywołanie tej samej funkcji *ShowArea()* dla dwóch różnych obiektów dało oczekiwany wynik w obu przypadkach. Gdy funkcja *ShowArea()* została wywołana z argumentem typu *Shape*, zostało policzone pole prostokąta. Z kolei, gdy wywołano *ShowArea()* dla obiektu klasy *Circle* (będącej pochodną klasy *Shape*), wywołano funkcję wirtualną z implementacją dla klasy *Circle*.

Dlaczego tak się stało?

Interesująca nas funkcja *Area()* dla której chcemy zastowować wielopostaciowość zadeklarowana jest wewnątrz klasy *Shape* jako wirtualna. Wewnątrz funkcji *ShowArea()* wywołujemy ją na rzecz referencji do obiektu typu bazowego. Zatem mechanizm polimorfizmu działa poprawnie.

Przykład 2

Kod

```cpp
#include <iostream>
#include <string>

// KLASA BAZOWA
class Plant
{
public:
    virtual void introduction() // funkcja wirtualna
    {
        std::cout << "Jestem roslinka" << std::endl;
    }
};

//KLASA POCHODNA
class Flower : public Plant
{
public:
    void introduction()
    {
        std::cout << "Jestem kwiatkiem" << std::endl;
    }
};

// KLASA POCHODNA
class Tree : public Plant
{
public:
    void introduction()
    {
        std::cout << "Jestem drzewem" << std::endl;
    }
};

int main()
{
    Plant plant;
    Flower flower;
    Tree tree;

    Plant* plantPtr = 0;    // wskaznik do obiektu typu bazowego

    plantPtr = &plant;
    plantPtr->introduction();

    plantPtr = &flower;
    plantPtr->introduction();

    plantPtr = &tree;
    plantPtr->introduction();

    return 0;
}
```

Widok konsoli

```
Jestem roslinka
Jestem kwiatkiem
Jestem drzewem
```

Wyjaśnienie

W klasie bazowej *Plant* znajduje się metoda wirtualna o nazwie *introduction*. Wypisuje na

ekranie pewną informację. Stworzono również dwie klasy pochodne: *Flower* oraz *Tree*. W funkcji *main()* stworzono obiekty wszystkich trzech typów oraz wskaźnik do typu bazowego *Plant*. Gdy ustawiono wskaźnik na obiekt plant i wywołano funkcję wirtualną *introduction()*, na ekranie widzimy tekst "Jestem roslinka", czyli ten z implementacji funkcji wirtualnej w klasie bazowej. Gdy przestawimy wskaźnik na obiekty klas pochodnych (*Flower* i *Tree*) i ponownie wywołamy *introduction()*, na ekranie zobaczymy teksty zaimplementowane w odpowiednich wersjach funkcji wirtualnej klas pochodnych.

Alternatywnie można zaimplementować metodę introduction() w klasie bazowej *Plant* jako czysto wirtualną, tj.:

```
virtual void introduction() = 0;
```

Należy jednak pamiętać, że nie można utworzyć wtedy obiektu typu *Plant* (oczywiście można, i nawet należy, utworzyć wskaźnik do typu *Plant*).

Dodatek

Operator rzutowania *dynamic_cast*

Główną rolę w polimorfizmie odgrywa fakt, że używamy wskaźnika (lub referencji) do typu bazowego i używamy go jakby był również do typów pochodnych. Bo czemu nie?
Każde skrzypce są szczególnym przypadkiem instrumentu, więc jeśli wskaźnik jest do instrumentu, może być tym bardziej do skrzypiec.
Można jednak za pomocą rzutowania sprawić, że wskaźnik do skrzypiec może pokazywać na instrument.

```
Violin* violin;
Intrument* instrument;
intrument = dynamic_cast<Violin*>(instrument);
```

Należy jednak pamiętać, że ta operacja nie zawsze ma sens, jednak ten operator ma to do siebie, że kompilator będzie czuwał nad poprawnością tej operacji.

Uwagi końcowe

1. Specyfikator *virtual* wystarczy zastosować tylko w klasie bazowej.
2. O tym, która wersja funkcji wirtualnej zostanie wywołana decyduje nie typ wskaźnika (referencji), ale to, na co pokazują w danym momencie.

Justyna Walkowiak