

Polimorfizm

Referat

Roman Toma

Wojciech Walczyszyn

Informatyka, Semestr 4, Grupa 4

Idea

Mechanizm pozwalający programiście na zdefiniowanie metod (metod wirtualnych) o tych samych nazwach, zarówno w klasie bazowej jak i pochodnej, a następnie wywoływanie tych metod w kodzie z poziomu tablicy wskaźników na klasę bazową jednocześnie dając gwarancję wywołania odpowiedniej metody (zdefiniowanej w klasie obiektu na rzecz, którego jest ona wywoływana). Daje to programiście łatwy sposób na rozszerzanie funkcjonalności swoich programów (konkretnie poprzez utworzenie nowej klasy pochodnej zawierającej implementację metody wirtualnej) bez większych zmian w kodzie. Ułatwia to również proces tworzenia oprogramowania – różne osoby mogą tworzyć odmienne implementacje klas pochodnych po wcześniejszym uzgodnieniu interfejsu (metod publicznych) klasy bazowej. Wadą polimorfizmu jest fakt, iż jego stosowanie zmniejsza czytelność kodu (zwiększa stopień abstrakcji) oraz jest bardziej zasobożerne (program musi przechowywać, podczas działania, informacje o tym jakiego typu jest klasa).

Więcej o idei polimorfizmu możemy przeczytać w:

<http://guidecpp.cal.pl/cplusplus.polimorph>

Sposób użycia

Przykładem polimorfizmu dynamicznego może być:

```
class Shape
{
public:
    virtual void drawShape();
};

class Triangle:
public Shape
{
public:
    void drawShape();
};

class Circle:
public Shape
{
public:
    void drawShape();
};

int main()
{
    Shape* tab[2];

    tab[0] = new Triangle;
    tab[1] = new Circle;

    tab[0]->drawShape(); //narysowano trójkąt
    tab[1]->drawShape(); //narysowano okrąg

    return 0;
}
```

W powyższym programie pomimo wywołania metody `drawShape()` na rzecz obiektu `tab[0]` (należy tutaj zauważyć, że tablica ta jest typu `Shape*`) zostanie wywołana metoda zaimplementowana w klasie `Triangle`. Dla obiektu `tab[1]` wywoływana jest metoda zaimplementowana w klasie `Circle`. Decyzję, którą metodę wykonać podejmuje program w trakcie działania.

Związki z dziedziczeniem

Polimorfizm pozwala na ujednoczenie interfejsów klasy bazowej i klas pochodnych poprzez stosowanie tych samych nazw metod, dając programiście możliwość rozszerzenia funkcjonalności klas bez poważnych zmian w kodzie klienta.

Obowiązkowe metody wirtualne

Aby skorzystać z polimorfizmu oferowanego przez C++ należy zadeklarować metodę jako wirtualną.

Metoda wirtualna pozwala na tzw. „późne wiązanie”, czyli podjęcie decyzji o tym, którą implementację metody wybrać w trakcie wykonywania się programu poprzez pobranie jej adresu z tablicy `VTABLE` (przechowuje wskaźniki do metod dla każdego obiektu). Skutkuje to jednak zwiększeniem wielkości obiektu.

Często stosuje się wirtualny destruktor, który zwalnia pamięć zaalokowaną przez obiekt klasy pochodnej, a następnie klasy bazowej (lub większej ich ilości). Destruktry wywołują się w odwrotnej kolejności niż zostały wywołane konstruktor.

Więcej o metodach wirtualnych pod linkiem w sekcji funkcje wirtualne:

<http://guidecpp.cal.pl/cplusplus/polimorph>

Metody i klasy abstrakcyjne

Klasa abstrakcyjna jest to taka klasa, której co najmniej jedna z metod jest czysto wirtualna. Ponadto nie można tworzyć obiektów tej klasy. Klasa abstrakcyjna jest swego rodzaju „prototypem” (szkieletem) dla klas pochodnych.

Przykład metody czysto wirtualnej:

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

Więcej o teorii klas abstrakcyjnych: http://4programmers.net/C/Klasy_Abstrakcyjne

Praktyczne przykłady można znaleźć na:

http://edu.pjwstk.edu.pl/wyklady/pro/scb/PRG2CPP_files/node131.html

Interfejsy

W języku programowania C++ interfejs może zostać zdefiniowany, jako klasa abstrakcyjna. Interfejs to taka klasa, która nie posiada żadnych danych, a jedynie prototypy metod. Chcąc stworzyć klasę implementującą interfejs, należy stworzyć klasę pochodną klasy abstrakcyjnej i zaimplementować wszystkie jej (klasy abstrakcyjnej) metody.

Więcej o interfejsach, czyli o abstrakcyjnych klasach bazowych można znaleźć pod poniższym linkiem w sekcji Abstract base classes: <http://www.cplusplus.com/doc/tutorial/polymorphism/>

Pytania i odpowiedzi:

1. Co należy zrobić aby skorzystać z polimorfizmu?

- a) Należy zadeklarować dwie metody o tej samej nazwie
- b) Należy zadeklarować metodę wirtualną
- c) Należy dołączyć plik nagłówkowy polymorphism.h

Odp.: b

2. Czy można utworzyć obiekt klasy abstrakcyjnej?

Odp.: Nie.

3. Jakie wymagania musi spełnić klasa, aby była interfejsem?

- a) Wystarczy, że posiada jedną metodę wirtualną, może posiadać pola.
- b) Musi posiadać zdefiniowane metody pozwalające na interakcję użytkownika z programem.
- c) Wszystkie jej metody muszą być wirtualne i bez implementacji, nie może posiadać pól.
- d) Musi być pustą klasą niemającą zadeklarowanych metod.

Odp.: c

4. Czym różni się metoda wirtualna od czysto wirtualnej?

- a) Niczym
- b) Metoda czysto wirtualna nie musi posiadać implementacji.
- c) Metoda czysto wirtualna nigdy nie przyjmuje żadnych argumentów

Odp.: b

5. Czy `virtual void drawShape() = 0;` jest równoważne: `virtual void drawShape() {}` ?

Odp.: Nie

6. Jakie możliwości daje nam wirtualny destruktor?

- a) Pozwala na utworzenie obiektu klasy pochodnej
- b) Umożliwia bezpośrednie wywołanie destruktora klasy pochodnej
- c) Wywołuje kaskadowo destruktory obiektów klas bazowych w kolejności odwrotnej do wywoływanych konstruktorów

7. Czy klasa:

```
class Employee
{
private:
    string name;
public:
    Employee();
    virtual int get_salary(int i);
};
```

jest klasą abstrakcyjną?

Odp.: Nie.

8. Czy poniższa deklaracja jest poprawna?

```
int show() = 0;
```

Odp.: Nie, ponieważ brakuje słowa kluczowego `virtual`

9. Co program pokaże na wyjściu:

```
class Shape
{
public:
    Shape();
    virtual void draw() { std::cout << "Jestem shape"; }
};
```

```
class Triangle : public Shape
{
public:
    Triangle();
```

```

        void draw() { std::cout << "Jestem triangle"; }
};

int main()
{
    Shape * a= new Triangle();
    a->draw();

    return 0;
}

```

Odp.: „Jestem triangle”

10. Czy klasa może implementować więcej niż jeden interfejs?

Odp.: Tak.

11. Zadeklaruj i zdefiniuj odpowiednie klasy, aby poniższy program:

```

int main()
{
    Samochod opel;
    Ciezarowka mann;
    Sportowy porche;

    Samochod* tir = &mann;
    Samochod* sport = &porche;

    opel.klakson();
    tir->klakson();
    sport->klakson();

    return 0;
}

```

Miał następujące działanie:

```

Jestem samochodem
Jestem ciezarowka
Jestem samochodem sportowym

```

Odp.:

```

class Samochod
{
public:
    virtual void klakson() { cout<<"Jestem samochodem\n"; }
};

class Ciezarowka:
public Samochod
{
public:
    void klakson() { cout<<"Jestem ciezarowka\n"; }
};

class Sportowy:
public Samochod
{
public:
    void klakson() { cout<<"Jestem samochodem sportowym\n"; }
};

```