Metody wirtualne

Paweł Nowik

13 stycznia 2020

Metoda wirtualna to metoda, której wywołanie jest polimorficzne. Polimorfizm jest to cecha programowania obiektowego, umożliwiająca różne zachowanie tych samych metod wirtualnych w czasie wykonywania programu. Oznacza to że można używać metod klasy pochodnej wszędzie tam, gdzie spodziewana jest klasa podstawowa. Metoda wirtualna nie może być zadeklarowana jako statyczna. Jeśli metoda wirtualna została zaimplementowana w jakimkolwiek wyższym stopniu dziedziczenia, nie jest konieczne podawanie implementacji w klasie pochodnej. W Javie domyślnie wszystkie metody są wirtualne, natomiast w C++ musimy użyć słowo kluczowe *virtual* w deklaracji metody.

Zastosowania

- Pozwala na rozszerzalność kodu. Zyskujemy możliwość płynnego rozwoju programów przez zastępowanie klas ich podklasami, co bez wirtualności jest niewykonalne.
- Programista nie musi przejmować się tym, którą z klas pochodnych aktualnie obsługuje, a jedynie tym, jakie operacje chce na tej klasie wykonać.

```cpp
#include <iostream>

const float pi = 3.14159;
class Figura {
  public:
    virtual float pole() const {
      return -1.0;
    }
};

class Kwadrat : public Figura {
  public:
    Kwadrat( const float bok ) : a( bok ) {}

    float pole() const {
      return a * a;
    }

  private:
    float a; // bok kwadratu
};

class Kolo : public Figura {
  public:
    Kolo( const float promien ) : r( promien ) {}

    float pole() const {
      return pi * r * r;
    }

  private:
    float r; // promien kola
};

void wyswietlPole( Figura& figura ) {
  std::cout << figura.pole() << std::endl;
  return;
}
```

```cpp
int main() {
  // deklaracje obiektow:
  Figura jakasFigura;
  Kwadrat jakisKwadrat( 5 );
  Kolo jakiesKolo( 3 );
  Figura* wskJakasFigura = 0; // deklaracja wskaźnika

  // obiekty ------------------------------
  std::cout << jakasFigura.pole() << std::endl; // wynik: -1
  std::cout << jakisKwadrat.pole() << std::endl; // wynik: 25
  std::cout << jakiesKolo.pole() << std::endl; // wynik: 28.274...

  // wskazniki ----------------------------
  wskJakasFigura = &jakasFigura;
  std::cout << wskJakasFigura->pole() << std::endl; // wynik: -1
  wskJakasFigura = &jakisKwadrat;
  std::cout << wskJakasFigura->pole() << std::endl; // wynik: 25
  wskJakasFigura = &jakiesKolo;
  std::cout << wskJakasFigura->pole() << std::endl; // wynik: 28.274...

  // referencje ---------------------------
  wyswietlPole( jakasFigura ); // wynik: -1
  wyswietlPole( jakisKwadrat ); // wynik: 25
  wyswietlPole( jakiesKolo ); // wynik: 28.274...

  return 0;
}
```

W przykładzie znajdują się deklaracje 3 klas: `Figura`, `Kwadrat` i `Kolo`. W klasie `Figura` została zadeklarowana metoda wirtualna (słowo kluczowe *virtual*) `virtual float pole()`. Każda z klas pochodnych od klasy `Figura` ma zaimplementowane swoje metody `float pole()`. Następnie (w funkcji main) znajdują się deklaracje obiektów każdej z klas i wskaźnika mogącego pokazywać na obiekty klasy bazowej `Figura`.

Wywołanie metod składowych dla każdego z obiektów powoduje wykonanie metody odpowiedniej dla klasy danego obiektu. Następnie wskaźnikowi `wskJakasFigura` zostaje przypisany adres obiektu `jakasFigura` i zostaje wywołana metoda `float pole()`. Wynikiem jest *"-1"* zgodnie z treścią metody `float pole()` w klasie `Figura`. Następnie przypisujemy wskaźnikowi adres obiektu klasy Kwadrat - możemy tak zrobić ponieważ klasa `Kwadrat` jest klasą pochodną od klasy `Figura` - jest to tzw. rzutowanie w górę. Wywołanie teraz metody `float pole()` dla wskaznika nie spowoduje wykonania metody zgodnej z typem wskaźnika - który jest typu `Figura*` lecz zgodnie z aktualnie wskazywanym obiektem, a więc wykonana zostanie metoda `float pole()` z klasy `Kwadrat` (gdyż ostatnie przypisanie wskaźnikowi wartości przypisywało mu adres obiektu klasy `Kwadrat`). Analogiczna sytuacja dzieje się gdy przypiszemy wskaźnikowi adres obiektu klasy `Kolo`. Następnie zostaje wykonana funkcja `void wyswietlPole(Figura&)` która przyjmuje jako parametr obiekt klasy `Figura` przez referencję. Tutaj również zostały wykonane odpowiednie metody dla obiektów klas pochodnych a nie metoda zgodna z obiektem jaki jest zadeklarowany jako parametr funkcji czyli `float Figura::pole()`. Takie działanie jest spowodowane przez przyjmowanie obiektu klasy `Figura` przez referencję. Gdyby obiekty były przyjmowane przez wartość (parametr bez &) zostałaby wykonana 3 krotnie metoda `float Figura::pole()` i 3 krotnie wyświetlona wartość *-1*.