Programowanie Obiektowe

Marcin Kamil Bączyk

Wykład 2

15 października 2020

Treść dzisiejszego wykładu

- przykład
- pojęcie obieku
- dwa podejścia do wytwarzania oprogramowania
- podstawowa obsługa wejścia i wyjścia
- tworzenie i używanie nowych typów danych

przykład

Wskaźniki i referencje.

pojęcie obiektu

Obiekt

wyodrębniony element rzeczywistości, mający znaczenie w rozpatrywanym modelu. Obiekt – element – może być materialny lub abstrakcyjny.

źródło: https://pl.wikipedia.org/wiki/Obiekt

Obiekt

Podstawowe pojęcie wchodzące w skład paradygmatu programowania obiektowego w analizie i projektowaniu oprogramowania oraz w programowaniu.
Każdy obiekt ma trzy cechy:

- **tożsamość**, czyli cechę umożliwiającą jego identyfikację i odróżnienie od innych obiektów;
- **zachowanie**, czyli zestaw metod wykonujących operacje na tych danych;
- **stan**, czyli aktualny stan danych składowych.

źródło:
https://pl.wikipedia.org/wiki/Obiekt_(programowanie_obiektowe)

Programowanie obiektowe

W obiektowym paradygmacie projektowania program (podprogram) wyrażany jest za pomocą zbioru obiektów, które komunikują się między sobą w celu wykonania zadań.
Poszczególne obiekty modelują konkretne fragmenty rzeczywistości.

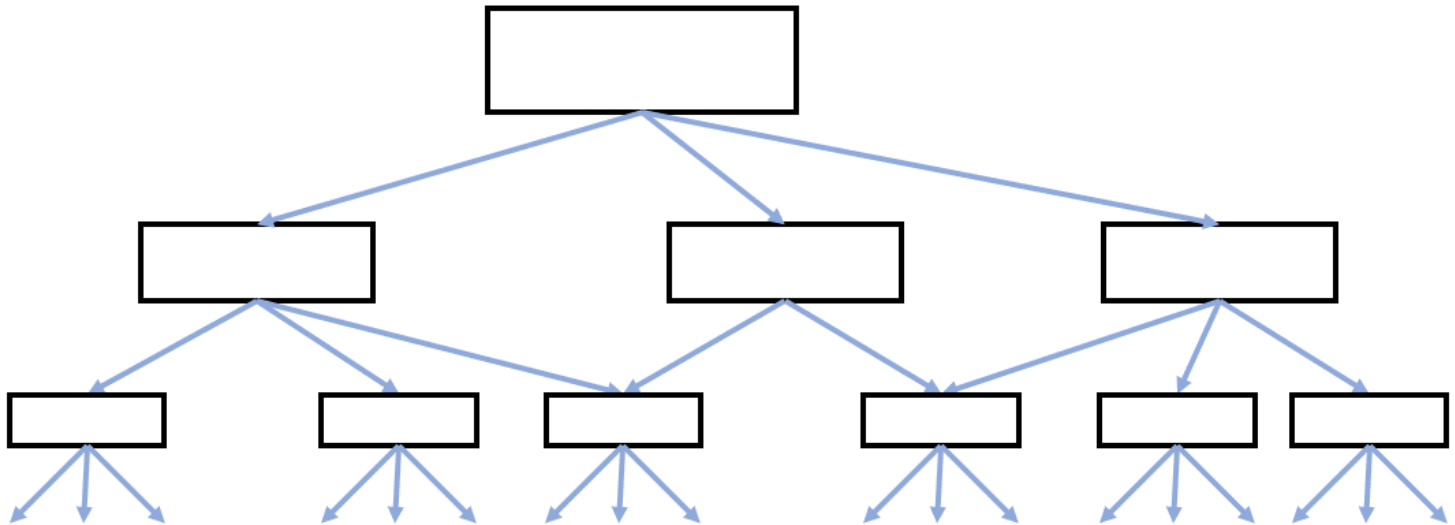
Cechy programowania obiektowego

- **abstrakcja**;
- **hermetyznacja**;
- **polimorfizm**;
- **dziedziczenie**.

dwa podejścia do wytwarzania oprogramowania

Podejście "z góry na dół" (top - down)

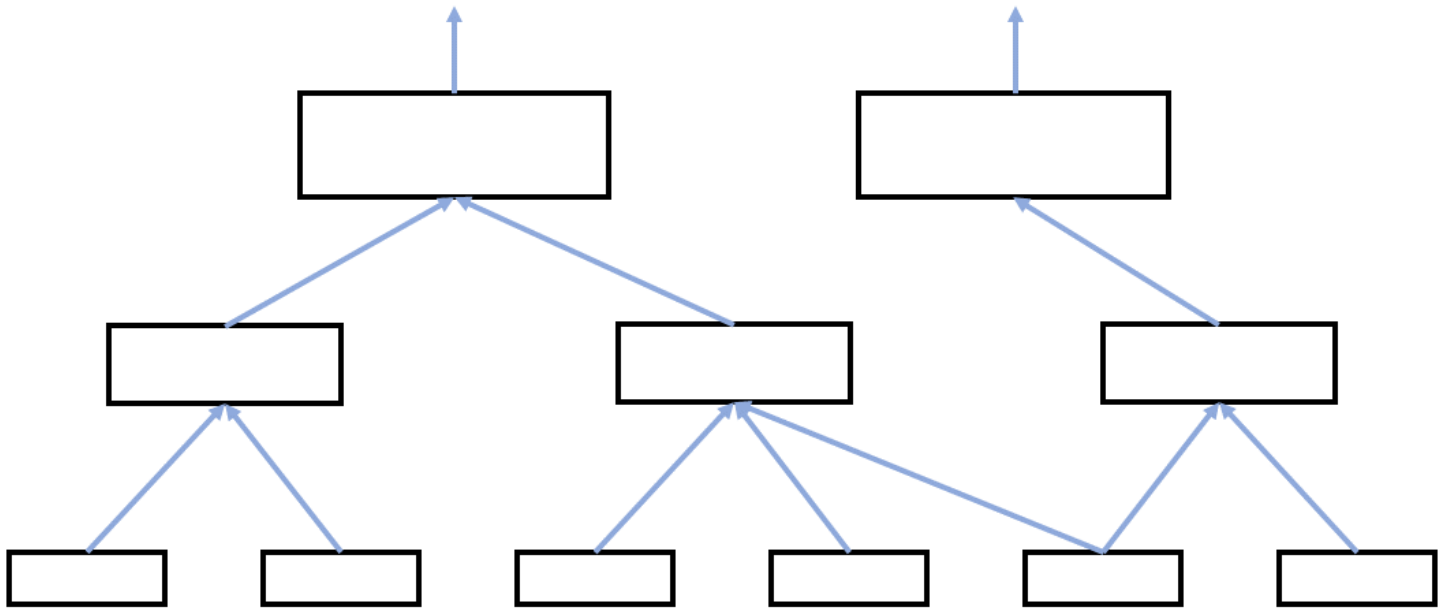
- Teoretycznie zgodne z paradygmatem strukturalnym.
- W pierwszej kolejności problem jest definiowany, zaś jego rozwiązanie delegowane jest do osobnych modułów.
- Stosowany częściej przy wytwarzaniu aplikacji.



dwa podejścia do wytwarzania oprogramowania

Podejście "z dołu do góry" (bottom - up)

- Teoretycznie zgodne z paradygmatem obiektowym.
- Definiowane są moduły, które mogą być wykorzystane do rozwiązania innych bardziej złożonych problemów.
- Stosowane raczej przy tworzeniu bibliotek.



podstawowa obsługa wejścia wyjścia

```
#include <iostream>
```

Obiekty globalny obsługujące strumienie

- std::cout - standardowe wyjście

```
std::cout << "standardowe wyjscie" << std::endl;
```

- std::cerr - standardowe wyjście dla błędów

```
std::cerr << "standardowe wyjscie bledow" << std::endl;
```

- std::clog - standardowe wyjście dla logów

```
std::clog << "standardowe wyjscie log" << std::endl;
```

- std::cin - standardowe wejście

```
int A;
std::cin >> A;
```

std::string - napisy w stylu C++

Język C++ udostępnia wygodny dla użytkownika typ danych reprezentujący ciągi znaków - std::string.

```
#include <string>

int main()
{
std::string s1 = "John␣Smith";
std::string s2 = "Ann␣Johnson";

std::cout << s1 << std::endl;
std::cout << s1 << "␣and␣" << s2 << std::endl;

return 0;
}
```

Więcej na :
http://www.cplusplus.com/reference/string/string/

std::string - napisy w stylu C++

Język C++ udostępnia wygodny typ do obsługi ciągów znakowych - std::string

https://en.cppreference.com/w/cpp/string/basic_string
http://www.cplusplus.com/reference/string/string/

```
int main(void){
    std::string s1("Napis_1");
    std::string s2 = "Napis_2";
    std::string s3 = s1;
    std::cout << s1 << "␣" << s2 << "␣" << s3 << std::endl;

    return 0;
}
```

std::string - napisy w stylu C++

Klasa std::string udostępnia kilka metod pozwalających na wygodne manipulowanie ciągami znaków, m.in.:

- lenght - zwraca długość napisu
- capacity - zwraca rozmiar przydzielonej pamięci
- clear - czyści napis
- find - przeszukuje napis w celu odnalezienia wzorca

```
std::string s_n("to jest napis");
std::string s_f("jest");
auto i = s_n.find(s_f);
```

- c_str - zwraca napis w stylu języka C
- operator[] - zwraca znak znajdujący się na wskazanym miejscu
- operator== i inne - porównywanie dwóch napisów

std::string - napisy w stylu C++

Moduł std::string udostępnia również kilka przydatnych funkcji, np.:

- stoi, ltoi, stof, ltod - konwersja napisu do wartości liczbowej

```
std::string s("11");
auto x = stoi(s);
```

- operator+ - łączy dwa napisy (konkatenacja)

```
std::string s1("Napis_1");
std::string s2 = "Napis_2";
auto s12 = s1 + s2;
```

- operator» - wczytywanie napisu ze strumienia
- operator« - wypisywanie napisu do strumienia

Tworzenie i używanie nowego typu danych

W budowanym systemie niezbędne jest narzędzie do zarządzania czasem. W trakcie implementacji kilku modułów wyższego rzędu wyodrębniono następujące brakujące funkcjonalności:

- pobieranie aktualnego czasu systemowego;
- porównywanie dwóch chwil czasowych i zwracanie różnicy wyrażonej w sekundach;
- wypisywanie na standardowym wyjściu zapisanego czasu w ustalonym (i niezmiennym) formacie (HH:MM:SS).

W jaki sposób należy przystąpić do projektowania modułu odpowiadającego za czas? Jak należy go zaimplementować?

Tworzenie i używanie nowego typu danych

Tworzenie nowego typu danych najlepiej rozpocząć od przygotowania krótkiego testu!

```
int main()
{
    auto t1 = now();

    wait(0.1);

    auto t2 = now();

    t1.print();
    t2.print();
    std::cout << t2.timeDifference(t1).value() << '\n';

    return 0;
}
```

Funkcja `now()` ma tworzyć obiekt reprezentujący czas zaś funkcja `wait(0.1)` ma zatrzymać program na 0.1 sekundy. *Implementacje tych metod dostępne w materiałach do wykładu.*

Tworzenie i używanie nowego typu danych

Definiując klasę najlepiej od razu zdefiniować typy pomocnicze. W tym przypadku `Duration` enkapsuluje dlugość interwału pomiędzy dwoma chwilami.

```
class Time
{
public:
    Time(short h, short m, float s);

    Duration timeDifference(const Time& t2) const;

    void print() const;

private:
    short h_ = 0;
    short m_ = 0;
    float s_ = 0;
};
```

W jaki inny sposób możemy przechowywać informację o czasie?

Tworzenie i używanie nowego typu danych

Definiując klasę najlepiej od razu zdefiniować typy pomocnicze. W tym przypadku `Duration` enkapsuluje dlugość interwału pomiędzy dwoma chwilami.

```
class Duration
{
public:
    Duration(float value);

    float value() const;
private:
    float value_;
};
```

Warto zwrócić uwagę, że dla tak zdefiniowanych klas nie istnieje możliwość zmiany stanu obiektu już po jego utworzeniu. Zapewnienie stałości obiektu na tym etapie jest często wykorzystywaną cechą. Dlaczego ?