Wrocław, 16 marca 2023

Paradygmaty Programowania Obiektowego

Laboratorium 2

Enkapsulacja (hermetyzacja)

prowadzący: *mgr inż. Marta Lampasiak*

1 Wprowadzenie – klasa, obiekt, enkapsulacja, hermetyzacja

Celem zajęć jest zapoznanie się z klasami, obiektami klas oraz z takimi zagadnieniami jak enkapsulacja i hermetyzacja. Klasa to inaczej mówiąc **typ**, ponieważ kreujemy nie jeden, konkretny obiekt, a **klasę** obiektów. Klasa składa się ze słowa kluczowego *class*, po którym występuje jej nazwa. Następnie w klamrach umieszczamy tzw. **ciało** klasy, czyli określenie z czego się składa. Posiada ona składowe w postaci danych (cech obiektów) oraz tzw. funkcje składowe (nazywane często **metodami**).

Klasa jest jak kapsuła, w której zamknięto dane oraz funkcje do posługiwania się nimi. Ta czynność zamknięcia nazywa się enkapsulacją („kapsułowaniem"). Jest to ważna cecha języka C++, ponieważ odzwierciedla nasze myślenie o obiektach. Obiekt typu czajnik to nie tylko jego składowe, czyli np. grzałka, ale też akcja jaką wykonuje, przykładowo funkcja *gotuj*.

Jeżeli klasa to nasza kapsuła, to pojęcie hermetyzacji można by metaforycznie rozumieć jako to, czy kapsuła jest zbudowana z przezroczystego materiału czy nie. Innymi słowy, czy coś może być dostępne spoza klasy czy nie. Wszystkie pola klasy są widoczne wewnątrz klasy. Oznacza to, że mają do nich dostęp funkcje składowe danej klasy. Mogą one jednak mieć różny poziom dostępności z zewnątrz, a więc z funkcji które nie są składowymi danej klasy: poziom dostępności jest określany jednym ze słów kluczowych: *public*, *private*, lub *protected*.

Poniżej znajduje się fragment kodu, który pokazuje definicję przykładowej klasy o nazwie *Osoba*. Zauważ, że funkcja składowa klasy ma dostęp do jej pól prywatnych i nie muszą być one przekazywane jako argumenty.

```
// ----- poczatek definicji klasy -----
class Osoba
{
  private:  // skladniki prywatne
    string nazwisko;
    int wiek;

  public:   // skladniki publiczne

    void zapamietaj(string, int); // deklaracja funkcji skladowej (metody) klasy

    // definicja funkcji skladowej w ciele klasy
    void wypisz()
    {
      cout << nazwisko << ", lat: " << wiek << endl;
    }
}
// ----- koniec definicji klasy -----

// definicja funkcji skladowej (metody) poza cialem klasy
// jest poza klasa, a wiec nazwa funkcji zostaje uzupelniona
// nazwa klasy i operatorem zakresu, czyli ::
void Osoba::zapamietaj(string napis, int lata)
{
  nazwisko = napis;
  wiek = lata;
}
```

Kolejnym zagadnieniem poruszanym na laboratorium jest specjalna funkcja składowa nazywana **konstruktorem**. Charakteryzuje się on tym, że nazywa się tak samo jak klasa. Nie posiada on żadnego określonego typu (nawet *void*). Konstruktor może być domyślny, czyli taki, który zostaje wywołany z pustą listą argumentów, ale mogą zostać utworzone również innego rodzaju konstruktory. Należy jednak pamiętać, że gdy tworzony jest obiekt danej klasy, wywoływana jest jedna implementacja konstruktora, którą kompilator może dopasować do listy parametrów inicjalizacyjnych obiektu, zgodnie z regułami przeciążenia. Jeżeli kompilator nie byłby w stanie dopasować żadnej definicji konstruktora, zgłoszony zostanie błąd w czasie kompilacji.

Specjalną funkcję składową klasy stanowi również **destruktor**. Wywoływany jest on automatycznie przed usunięciem obiektu z pamięci. Jego działanie jest zatem w pewnym sensie przeciwstawne do działania konstruktora. Destruktor nie może ulegać przeciążeniom.

2 Zadania

1. *(klasy, obiekty, hermetyzacja)* (*Po wykonaniu tego i kolejnego zadania otrzymuje się* ocenę 3.0. Wykonaj następujące polecenia:

   - Napisz klasę `Komiks` zawierającą **publiczne** pola: `tytul`, `autor`, `wydawca`.
   - Napisz **funkcję** o nazwie `wypisz`, która jako argument otrzymuje obiekt tej klasy i wypisuje na standardowym wyjściu wartości pól otrzymanego w argumencie obiektu.
   - Napisz **funkcję** `wczytaj`, która jako argument otrzymuje **referencję** do obiektu i wczytuje ze standardowego wejścia wartości pól obiektu, do którego referencje otrzymała w argumencie. Zauważ, że może wystąpić potrzeba wczytania napisu kilkuczłonowego, czyli razem ze spacją. Przygotuj swój program na tego rodzaju przypadek.
   - Dopisz **do klasy metody**: wczytującą (ze standardowego wejścia dane dla pól obiektu) i wypisującą pola obiektu.
   - Zaprezentuj działanie utworzonych funkcji tworząc **obiekty statyczne**.
   - Zmień dostęp tylko do pól klasy z publicznego na prywatny. Funkcje składowe (metody) niech pozostaną publiczne. Przetestuj i odpowiedz na pytanie w komentarzu w kodzie: Czy stworzone przez Ciebie funkcje składowe i funkcje nie będące składowymi klasy będą nadal działać? Jeśli tak, to które i dlaczego?

2. *(hermetyzacja, konstruktor, destruktor, lista inicjalizacyjna, funkcje dostępowe)* Utwórz klasę o nazwie `Pokoj` posiadającą składowe **prywatne**:

   - `nazwa` – zmienna typu `string`
   - `dlugosc` – zmienna typu `float`
   - `wysokosc` – zmienna typu `float`,
   - `szerokosc` – zmienna typu `float`.

   Następnie wykonaj poniższe kroki:

   - Utwórz konstruktor wieloargumentowy, który podczas tworzenia obiektu nada polom klasy wartości przekazane w argumentach. Definicję konstruktora umieść poza klasą. Dodatkowo w konstruktorze wyświetl na standardowym wyjściu tekst: *Nadanie wartosci – konstruktor wieloargumentowy.*
   - Utwórz destruktor, który w swoim ciele jedynie wyświetla na standardowym wyjściu tekst: *Likwiduje!.*
   - Wywołaj konstruktor dla stworzonego obiektu statycznego.

3. *Po wykonaniu tego zadania uzyskuje się ocenę 4.0*. Kontynuuj tworzenie programu z poprzedniego zadania, na pytania odpowiedz w komentarzach w kodzie:

   - Dokonaj inicjalizacji pól w klasie danymi: *Sypialnia*, *4.5*, *2.5*, *2.5*.
   - Utwórz metodę o nazwie `wyswietl_dane`, która pozwoli wyświetlić na standardowym wyjściu wartości obiektu.
   - Wyświetl dane utworzonego w poprzednim zadaniu obiektu statycznego poprzez utworzoną metodę. Zaobserwuj działanie programu.
   - Utwórz konstruktor domyślny, który jedynie wyświetla napis: *Nadanie wartosci – konstruktor domyslny*, a poza tym nic nie robi. Następnie utwórz obiekt statyczny i wyświetl jego dane, czyli zaprezentuj działanie konstruktora domyślnego.
   - Jakie wartości zostały wyświetlone dla stworzonego obiektu?
   - Czy dla utworzonej przez Ciebie klasy było konieczne zdefiniowanie konstruktora domyślnego jawnie, aby mógł on zostać wywołany?
   - Czy dla stworzonych obiektów można zauważyć działanie destruktora bez jego jawnego wywołania na rzecz obiektu? Dlaczego tak się dzieje?

4. *Po wykonaniu tego zadania uzyskuje się ocenę 5.0*. Kontynuuj program z poprzedniego zadania, na pytania odpowiedz w komentarzach w kodzie:

   - Skopiuj kod konstruktora wieloargumentowego w celu jego modyfikacji. Pozostaw jednak dotychczasową implementację zakomentowaną. Przerób konstruktor w taki sposób, aby wykorzystać w nim listę inicjalizacyjną. Czy uzyskany efekt różni się od tego, jaki uzyskaliśmy przy poprzedniej implementacji konstruktora?
   - W klasie możemy zdefiniować tzw. funkcje modyfikujące i funkcje dostępowe. Te drugie powinny zostać zdefiniowane ze słowem kluczowym *const*. Umieść je przy odpowiednich, stworzonych w tym zadaniu przez siebie funkcjach.
   - Wyjaśnij, dlaczego akurat przy tych funkcjach użyłeś słowa kluczowego *const*?
   - Ile destruktorów może mieć klasa?