Opracowanie: dr inż. Łukasz Maliński i mgr inż. Błażej Nycz, Politechnika Śląska, 2022

LABORATORIUM 2.1
KLASY I ENKAPSULACJA

Poruszane zagadnienia z dziedziny programowania:

- Klasa i instancja klasy - **wykład 5,** *„Opus Magnum C++11”*
- Pola i metody klasy - **wykład 5,** *„Opus Magnum C++11”*
- Enkapsulacja i interfejs klasy - **wykład 5,** *„Opus Magnum C++11”*
- Metody użytkowe klasy

Umiejętności do opanowania:

- definiowanie klasy zgodnie z zasadą enkapsulacji i dobór pól na bazie opisu słownego,
- definiowanie interfejsu klasy zgodnie z zadanymi wytycznymi, oraz pisanie kodu nakładającego ograniczenia na wartości pól.
- definiowanie prostych metod użytkowych klasy.
- odczyt i modyfikacja stanu pojedynczego bitu w zmiennej,
- projektowanie struktury wewnętrznej klasy, tak aby uzyskać logiczny podział na funkcjonalności wewnętrzne i zewnętrzne, z uwzględnieniem zasady DRY.

Oznaczenia odnośnie samodzielności pracy:

■ – **Ten problem koniecznie rozwiąż w pełni samodzielnie.** Możesz posługiwać się literaturą, wykładami i dokumentacją w celu sprawdzenia niuansów składniowych, ale nie szukaj gotowych rozwiązań i algorytmów. **Jest to bardzo ważne z punktu widzenia nauki i oszukiwanie przyniesie tylko poważne braki w późniejszych etapach nauki!**

▲ – Rozwiązując ten problem możesz posiłkować się rozwiązaniami zapożyczonymi od innych programistów, **ale koniecznie udokumentuj w kodzie źródło.** Nie kopiuj kodu bezmyślnie, tylko dostosuj go do kontekstu zadania. **Koniecznie musisz dokładnie zrozumieć rozwiązanie, które adoptujesz, gdyż inaczej wiele się nie nauczysz!**

● – Rozwianie tego problemu właśnie polega na znalezieniu i użyciu gotowego rozwiązania, **ale koniecznie podaj źródło.** Nie będzie wielkiej tragedii jeśli wykażesz się tylko ograniczonym zrozumieniem rozwiązania, gdyż nie musi być ono trywialne.

Przykładowe zadania do rozwiązania w ramach samodzielnej nauki (nie oceniane):

Zadanie A (z rozwiązaniem):

▲ Stosując się do zasady enkapsulacji napisz klasę **Książka**, która przechowa informacje takie jak: autor, tytuł, liczba stron, rok wydania. Samodzielnie dobierz typy pól.

Wskazówki: Przyjmij jedną stałą uniwersalną długość dla informacji tekstowej i koniecznie używaj w programie zwykłych tablic znakowych (nie std::string). Imię i nazwisko autora traktuj jako jeden tekst. Zauważ, że zarówno liczba stron jak i rok wydania to są zawsze liczby całkowite.

▲ Zdefiniuj dla klasy **Książka** podstawowe akcesory - setery i getery dla wszystkich pól. W seterach nie pozwól na zadanie liczby stron mniejszej niż 20, ani roku wydania sprzed 1500 roku.

Wskazówki: Tak napisz getery, do pól tekstowych, aby możliwy był bezpośredni odczyt zawartości tablicy tekstowej, ale nie można było jej zmodyfikować przy pomocy tego getera. W seterach pól tekstowych nie pozwól na przekroczenie rozmiaru tablic znakowych przewidzianych na przechowywanie informacji. Jest to bardzo ważne, gdyż przekroczenie tablicy może uszkodzić wartości innych pól w klasie. Pisząc setery do pól liczbowych uwzględnij zadane ograniczenia. Jeśli przekazana do zapisu wartość (argument setera), nie spełnia ograniczenia, to ustaw pole na najbliższą poprawną wartość.

Przykładowe rozwiązanie:

```
constexpr size_t TEXT_LENGHT = 100;
using tekst = char[TEXT_LENGHT];

class Ksiazka
{
private:
  // Pola Klasy:
  size_t m_liczbaStron, m_rokWydania;
  tekst m_autor, m_tytul;
public:
  // Setery:
  void setLiczbaStron(int i_number)
  {
    m_liczbaStron = (i_number > 20) ? i_number : 20;
  }
  void setRokWydania(int i_number)
  {
    m_rokWydania = (i_number > 1500) ? i_number : 1500;
  }
  void setAutor(const char* i_autor)
  {
    strncpy(m_autor, i_autor, TEXT_LENGHT);
  }
  void setTytul(const char* i_tytul)
  {
    strncpy(m_tytul, i_tytul, TEXT_LENGHT);
  }
  // Getery:
  /* Zauważ, że getery  nazw zwracają const char*, aby nie dało się tym geterem modyfikować
     treści! */

  size_t getLiczbaStron() { return m_liczbaStron };
  size_t getRokWydania() { return m_rokWydania; }
  const char* getAutor() { return m_autor; }
  const char* getTytul() { return m_tytul; }
};
```

Zadanie B (z rozwiązaniem):

▲ Stosując się do zasady enkapsulacji i zasad podziału na pliki, napisz klasę **Student**, która przechowa informacje takie jak:

- imię,
- nazwisko,
- semestr studiów (liczba naturalna 1 do 8),
- stopień studiów (np.: I, II ,III),
- nr albumu (liczba naturalna, która zawsze ma dokładnie 6 cyfr)
- kierunek,
- płeć.

Dla pól: stopień studiów, płeć i kierunek utwórz odpowiednie typy wyliczeniowe, tak aby kierunek można było wybrać z przynajmniej 3 różnych opcji.

▲ Zdefiniuj wszystkie podstawowe akcesory (setery i getery), oraz napisz jeden seter zbiorczy, pozwalający zadawać wszystkie dane na raz (staraj się stosować do zasady DRY).

Wskazówki: Zauważ, że semestr powinien być liczbą z przedziału <1-8> (dla stopnia I i III) i <1-4> (dla stopnia II), a numer albumu powinien być zawsze liczbą 6 cyfrową (np.: numer 123 należy przechowywać i jako 000123).

■ Napisz metodę użytkową podającą, na którym roku jest aktualnie student uwzględniając numer semestru i stopień studiowania. Rok licz od rozpoczęcia I stopnia studiów.

Wskazówki: Dla uproszczenia przyjmij, że studia I stopnia trwają dokładnie 4 lata, II stopnia 2 lata i III stopnia maks. 4 lata. Jeśli student jest na II stopniu to znaczy, że ma zaliczone 4 lata studiów, więc jest na 5 roku (jeśli semestr to 1 lub 2), lub na 6 roku (jeśli semestr to 3 lub 4). Aby dostać się na studia III stopnia (doktoranckie) należy mieć zaliczony I i II stopień.

Przykładowe rozwiązanie:

Plik student.h:

```
#pragma once
#include<cstring>

constexpr size_t TEXT_LENGHT = 60;
using tekst = char[TEXT_LENGHT];
using nrAlbumu = char[7];
enum class stopien { pierwszy, drugi, trzeci };
enum class kierunek { prawo, filozofia, informatyka };
enum class plec {kobieta, mezczyzna };

/* Zauważ, do opisu numeru albumu wykorzystano tablicę znakową o 7 elementach (6 na cy-
fry numeru i 1 na znak NUL. Takie rozwiązanie jest wygodne do zapisu liczb, które zawsze
mają stałą liczbę cyfr. */

class Student
{
  // Pola:
  tekst m_Imie, m_Nazwisko;
  unsigned short m_Semestr;
  stopien m_stopienStudiow;
  nrAlbumu m_NrAlbumu;
  kierunek m_Kierunek;
public:
  // Getery:
  const char* getImie() { return m_Imie; }
  const char* getNazwisko() { return m_Nazwisko; }
  unsigned short getSemestr() { return m_Semestr; }
  stopien getStopienStudiow() { return m_stopienStudiow; }
  const char* getNrAlbumu() { return m_NrAlbumu; }
  kierunek getKierunek() { return m_Kierunek; }
  // Setery:
  void setImie(const char* i_Imie);
  void setNazwisko(const char* i_Nazwisko);
  void setSemestr( short i_Semestr);
  void setStopienStudiow( stopien i_Stopien);
  void setNrAlbumu(const char* i_NrAlbumu);
  void setKierunek( kierunek i_Kierunek);
  // Seter zbiorczy:
  void setAll(const char* i_Imie, const char* i_Nazwisko, short i_Semestr,
              stopien i_Stopien, const char* i_NrAlbumu,  kierunek i_Kierunek);
  // Metoda użytkowa:
  unsigned short ktoryRok();
};
```

Plik student.cpp:

```
void Student::setImie(const char* i_Imie)
{
  strncpy(m_Imie, i_Imie, TEXT_LENGHT);
}
void Student::setNazwisko(const char* i_Nazwisko)
{
  strncpy(m_Nazwisko, i_Nazwisko, TEXT_LENGHT);
}
void Student::setSemestr( short i_Semestr)
{
```

```
  m_Semestr = i_Semestr;
}
void Student::setStopienStudiow( stopien i_Stopien)
{
  m_stopienStudiow = i_Stopien;
}
void Student::setNrAlbumu( const char* i_NrAlbumu)
{
  strncpy(m_NrAlbumu, i_NrAlbumu, 6);
}
void Student::setKierunek( kierunek i_Kierunek)
{
  m_Kierunek = i_Kierunek;
}
void Student::setAll(const char* i_Imie, const char* i_Nazwisko, short i_Semestr,
                     stopien i_Stopien, const char* i_NrAlbumu, kierunek i_Kierunek)
{
    setImie(i_Imie);
    setNazwisko(i_Nazwisko);
    setSemestr(i_Semestr);
    setStopienStudiow(i_Stopien);
    setNrAlbumu(i_NrAlbumu);
    setKierunek(i_Kierunek);
}
unsigned short Student::ktoryRok()
{
   return 1 + (m_Semestr-1) / 2 + (m_stopienStudiow == stopien::drugi) * 4
                                + (m_stopienStudiow == stopien::trzeci) * 6;
}
```

```
/* W tej metodzie, zastosowano bardzo sprytny trik. Jeśli ktoś jest na II stopniu studiów, to działanie (m_stopienStudiow == stopien::drugi) da wynik true, który intepretowany jest jako liczba 1. Pozwala to opcjonalnie dodać 4 do wyniku pierwszego członu. Gdy ktoś jest na III stopniu to znaczy że ma ukończone 6 lat studiów + tyle ile wynika z aktualnego semestru. Nie można być jednocześnie na II i III stopniu (przynajmniej w naszym programie), więc albo dodajemy 4 albo 6 lat. */
```

Zadania domowe (oceniane)

UWAGA: Można oddać tylko jedno zadanie domowe. Ocena za zadanie domowe jest uwzględniana tylko jeśli uzyska się przynajmniej 1,0 pkt za rozwiązanie zadań podanych na zajęciach!

W żadnym z zadań nie można używać std::string!

Zadanie 1 [1,0 pkt]:

- Zdefiniuj klasę Fotografia, przechowującą informację o:
  - autorze (wskazanie innego typu złożony w programie),
  - wymiarach (szerokość i wysokość w pikselach,
  - ścieżce do pliku graficznego na dysku twardym.

Samodzielnie dobierz właściwe typy pól.

Wskazówki: Przyjmij ze autor to jest pusta Struktura zdefiniowana w programie. Jej zawartość nie jest istotna, ważne jest żeby można było utworzyć jej instancję i doczytać adres tej instancji. Pole w klasie powinni przechowywać gdzie w pamięci znajduje się autor. Wymiary powinny być liczbami naturalnymi.

- Dla wszystkich pól klasy zdefiniuj setery i getery, przy czym oba wymiary zadawaj jednym wspólnym seterem (odczyt dalej wymaga dwóch geterów). W seterze rozmiarów uwzględnij ograniczenia.

Wskazówki: Zauważ, że wymiary obrazu powinny być liczbami naturalnymi. Nie pozwól na wprowadzenie niepoprawnych wymiarów do instancji i w takim przypadku przyjmij najmniejsze poprawne wartości.

■ Napisz metodę użytkową obliczającą rozmiar pamięci potrzebnej na przechowanie fotografii kolorowej. Przyjmij, że jeden piksel zajmuje w pamięci 4 bajty. Metoda powinna zwracać wynik w bajtach (jeśli rozmiar jest mniejszy niż 200 bajtów), w kilobajtach jeśli rozmiar jest większy od 200 bajtów i mniejszy o 200 kB), i w megabajtach w każdym innym przypadku.

Ocenianie:

Dobór pól klasy:	0,2 pkt.
Interfejs klasy:	0,2 pkt.
Ograniczenia:	0,2 pkt.
Metoda użytkowa:	0,2 pkt.
Ogólna jakość kodu:	0,2 pkt.

Zadanie 2 [1,0 pkt]:

■ Zdefiniuj klasę Lek, która zwierać będzie informacje o:

- nazwie leku,
- cenie (w PLN ale uwzględniaj dokładność przynajmniej do 1 grosza),
- stanie dostępności (na receptę, ogólnodostępny, niedostępny),
- zamienniku (wskazanie innego leku).

Samodzielnie dobierz właściwe typy pól.

Wskazówki: Zauważ, że stan dostępności to wybór jednej z trzech opisowych opcji. Zamiennik powinien wskazywać inny lek istniejący w programie, który można zaproponować jako zamiennik.

■ Dla wszystkich pól klasy zdefiniuj setery i getery, W seterze ceny uwzględnij ograniczenie, że cena nie może być niższa nić 4,99 PLN. W geterze ceny uwzględnij że leki na receptę są czyściwo refundowane (50% upustu).

Wskazówki: Przyjmij że cena to zawsze jest cena rynkowa leku Dla uproszczenia nie doliczaj podatków i marży apteki. Jeśli dany lek jest na receptę to zwróć cenę skorygowaną o wartość upusty. W innym wypadku zwróć faktyczną zapisana cenę.

■ Napisz metodę użytkową, która przyjmie wartość rabatu (wyrażony w procentach) i zwróci wskazanie do leku lub zamiennika, w zależności, który okaże się tańszy. Przyjmij, że rabat dotyczy tylko leku dla którego wywołano metodę.

Wskazówki: Zauważ, że metoda ma zwrócić wskazanie (adres) leku. Jeśli lek, dla którego ją wywołano po naliczeniu rabatu okaże się tańszy od zamiennika (w oryginalnej cenie), to należy zwróci adres instancji tego leku. W innym wypadku adres instancji zamiennika.

Ocenianie:

Dobór pól klasy:	0,2 pkt.
Interfejs klasy:	0,2 pkt.
Ograniczenia:	0,2 pkt.
Metoda użytkowa:	0,2 pkt.
Ogólna jakość kodu:	0,2 pkt.

Zadanie 3 [2,0 pkt]:

■ Samodzielnie zaprojektuj i zdefiniuj klasę do opisu **Generatora** liczb losowych, która pozwoli na:

losowanie liczb całkowitych z zadanego przedziału,

- losowanie liczb rzeczywistych z zadanego przedziału, z możliwością wskazania, czy poszczególne granice są otwarte, czy zamknięte,
- to by wylosowana wartość nie powtórzyła się w 3 kolejnych losowaniach,
- zadanie ziarna generatora na korektą wartość lub inicjalizację zegarem systemowym.

Wskazówki: Wszystkie powyższe wymagania powinny być zrealizowane jako oddzielne metody użytkowe klasy.

- Granice losowania jak i stany otwarcia/zamknięcia granic, nie mogą być podawane z zewnątrz przy każdym losowaniu - powinno dać się wygenerować dowolną liczbę wartości losowych bez każdorazowego podawania granic. Nie pisz osobnych metod publicznych dla każdej konfiguracji otwarcia/zamknięcia granic.

Wskazówki: Przemyśl dobrze, jakie będą pola w klasie i jakie akcesory do nich utworzysz. Zastanów się, czy wszystkie pola będą musiały mieć akcesory. W klasie aż się prosi o pola których w żaden sposób nie będzie widać z punktu widzenia interfejsu klasy.

- Tam gdzie to możliwe staraj się stosować do zasady DRY i pewne wspólne funkcjonalności udostępniaj tylko wewnątrz klasy jako narzędzia wewnętrzne.

Wskazówki: Zauważ, że przy losowaniu liczb rzeczywistych, przeskalowanie zakresu z 0-1 do a-b, to operacja, która jest identyczna, bez względu na stan otwarcia granic.

- Zaprezentuj działanie klasy **Generatora** w programie głównym.

Ocenianie:

Projekt klas, dobór pól i akcesory:	0,6 pkt.
Działanie metod użytkowych:	0,4 pkt.
Blokada losowania tych samych wartości:	0,4 pkt.
Przestrzeganie zasady DRY:	0,2 pkt.
Prezentacja w programie głównym:	0,2 pkt.
Ogólna jakość kodu:	0,2 pkt.