Book – część II

Zadanie (przypomnienie)

Chcemy opracować aplikację katalogu książek, która przechowuje listę książek i umożliwia użytkownikowi wyszukiwanie książek, które odpowiadają atrybutom docelowym użytkownika.

- Użytkownik musi mieć możliwość dodawania książek i ich atrybutów do katalogu.
- Atrybutami każdej książki będą tytuł książki oraz nazwisko i imię autora.

Zadanie – ciąg dalszy

Załóżmy teraz, że mamy dodatkowe wymagania, aby dodać dwa kolejne atrybuty książki — rok publikacji i gatunek:

- Atrybuty dla każdej książki beletrystycznej powinny obejmować rok publikacji i gatunek.
- Gatunkami książek powinny być ADVENTURE, CLASSICS, DETECTIVE, FANTASY, HISTORIC, HORROR, ROMANCE, and SCIFI.

Rozwiązanie

Nasuwa się pomysł zdefiniowania dwóch kolejnych atrybutów w klasie Book:

- year,
- genre.

Rozwiązanie

Wszelkie zmiany atrybutów, które wprowadzamy do klasy Book, będą wymagały odpowiednich zmian w klasie `Catalogue`:

- Metoda `add()` ma atrybuty jako parametry.
- Instrukcja if w metodzie `find()` odwołuje się do atrybutów

Rozwiązanie

Zmiany w klasie Book „przeciekają” do klasy Catalogue

The Encapsulate What Varies Principle
Zasada enkapsulacji tego, co się zmienia

- Dobry projekt oprogramowania oddziela kod, który może się zmieniać, od kodu, który się nie zmienia.
- Enkapsulacja kodu, który może się zmieniać, izoluje go od reszty programu. Następnie, gdy w enkapsulowanym kodzie zajdą zmiany, zmiany te nie wyciekną i nie spowodują zmiany w innym kodzie.
- Typowym sposobem enkapsulacji kodu, który może się zmieniać, jest umieszczenie go w osobnej klasie.

Klasa Attributes

- Kluczem będzie zdefiniowanie klasy `Attributes`.
- Spowoduje to przebudowanie aplikacji, ale za to dalsze dodawanie kolejnych atrybutów stanie się łatwe.

class enum

```
enum class Genre
{
    ADVENTURE, CLASSICS, DETECTIVE, FANTASY, HISTORIC,
    HORROR, ROMANCE, SCIFI, UNSPECIFIED
};
```

Lepiej użyć enum niż string

```
std::ostream& operator<<(std::ostream& ostr, const Genre& genre)
{
    switch (genre)
    {
        case Genre::ADVENTURE: ostr << "adventure"; break;
        case Genre::CLASSICS:  ostr << "classics";  break;
        case Genre::DETECTIVE: ostr << "detective"; break;
        case Genre::FANTASY:   ostr << "fantasy";   break;
        case Genre::HISTORIC:  ostr << "historic";  break;
        case Genre::HORROR:    ostr << "horror";    break;
        case Genre::ROMANCE:   ostr << "romance";   break;
        case Genre::SCIFI:     ostr << "scifi";     break;
        default: ostr << "unspecified"; break;
    }
    return ostr;
}
```

```
class Attributes {
private:
    std::string title, last, first; // stare atrybuty
    int    year;                    // nowy atrybut
    Genre  genre;                   // nowy atrybut
    static bool equal_ignore_case(const std::string& string1, const
std::string& string2); // przeniesione z klasy Catalogue
public:
    Attributes(const std::string& ttl,
               const std::string& lst,
               const std::string& fst,
               int yr, Genre gen)
        : title(ttl), last(lst), first(fst), year(yr), genre(gen) {}
    // gettery pominięte dla zaoszczędzenia miejsca na slajdzie
    bool is_match(const Attributes& target_attrs) const;
};
```

Implementacja metody is_match

```
bool Attributes::is_match(const Attributes& target_attrs) const
{
    return equal_ignore_case(target_attrs.get_title(), title)
           && equal_ignore_case(target_attrs.get_last(), last)
           && equal_ignore_case(target_attrs.get_first(), first)
           && (    (target_attrs.get_year() == 0)
                ||  (target_attrs.get_year() == year))
           && (    (target_attrs.get_genre() == Genre::UNSPECIFIED)
                ||  (target_attrs.get_genre() == genre));
}
```

Klasa Book

```
#include "attributes.h"
class Book
{
private:
    Attributes* attributes;
public:
    Book(Attributes* attrs) : attributes(attrs) {}
    Attributes* get_attributes() const {return attributes;}
};
```

Klasa Catalogue

```
#include "Book.h"
class Catalogue
{
private:
    std::vector<Book *> booklist;
public:
    void add(Attributes* attrs);
    std::vector<Book *> find(const Attributes& target_attrs) const;
};
```

Klasa Catalogue – implementacja

```
void Catalogue::add(Attributes* attrs)
{
    booklist.push_back(new Book(attrs));
}
```

Klasa Catalogue – implementacja

```
vector<Book *>Catalogue::find(const Attributes& target_attrs) const
{
    vector<Book *> matches;
    for (Book *book : booklist)
    {
        Attributes *book_attrs = book->get_attributes();
        if (book_attrs->is_match(target_attrs))
                matches.push_back(book);
    }
    return matches;
}
```

Koniec

Po zmianie kodu testującego (plik main.cpp jest zamieszczony na stronie) wszystko powinno działać

Podsumowanie

- Czy to koniec?
- W tej chwili dodajemy do katalogu atrybuty, chyba jednak powinniśmy dodawać książki
- ***Potrzeba wielu iteracji, żeby stworzyć dobry projekt***