PROGRAMOWANIE OBIEKTOWE – LABORATORIUM

ĆWICZENIE 5 – PRZETWARZANIE PLIKÓW II

---

1. Cel ćwiczenia

Zapoznanie się z mechanizmem dziedziczenia i zastosowaniem polimorfizmu w języku C++. Do realizacji tego ćwiczenia wymagany jest kod z ćwiczenia 2.

2. Pojęcia

Dziedziczenie - mechanizm współdzielenia funkcjonalności między klasami. Klasa może dziedziczyć po innej klasie, co oznacza, że oprócz swoich własnych pól oraz metod, uzyskuje także te pochodzące z klasy, z której dziedziczy, pod warunkiem, że są one określone odpowiednim specyfikatorem dostępu. Klasa dziedzicząca jest nazywana klasą pochodną lub potomną (w j. angielskim: *subclass* lub *derived class*), zaś klasa, z której następuje dziedziczenie — klasą bazową (w ang. *superclass*). Z jednej klasy bazowej można uzyskać dowolną liczbę klas pochodnych. Klasy pochodne posiadają obok swoich własnych metod i pól, również kompletny interfejs klasy bazowej.

Polimorfizm dynamiczny - Referencje i wskaźniki do obiektów mogą dotyczyć obiektów różnego typu, a wywołanie metody dla referencji spowoduje zachowanie odpowiednie dla pełnego typu obiektu wywoływanego. Innymi objawami polimorfizmu są m.in. przeciążanie metod i operatorów.

Metoda wirtualna - metoda, których zachowanie może zostać nadpisane w klasach pochodnych. Jej wywołanie uwzględnia rzeczywisty typ obiektu, w którym została wywołana.

Wzorzec projektowy - uniwersalne, sprawdzone w praktyce rozwiązanie często pojawiających się, powtarzalnych problemów projektowych. Pokazuje powiązania i zależności pomiędzy klasami oraz obiektami i ułatwia tworzenie, modyfikację oraz pielęgnację kodu źródłowego.

Fabryka - kreacyjny wzorzec projektowy, którego celem jest dostarczenie interfejsu do tworzenia obiektów.

Pojęcia do zapoznania przed laboratorium: wirtualny destruktor**,** metoda czysto wirtualna, klasa abstrakcyjna, metody statyczne, dynamiczna alokacja pamięci.

3. Instrukcja laboratoryjna

a. Stworzyć abstrakcyjną klasę *IFile*, zawrzeć w niej metody z klas *CsvFile* i *BinaryFile*: *Read(vector<Point>&), Read(Point&, int), Write(vector<Point>&)*. Metody w klasie *IFile* muszą być czysto wirtualne.
b. Przenieść prywatne pola z klas *CsvFile* i *BinaryFile* do klasy *IFile*.

c. Zmienić deklaracje klas *CsvFile* i *BinaryFile*, tak aby dziedziczyły po klasie *IFile*.
d. Stworzyć klasę *FileFactory*, która będzie zawierać tylko jedną metodę statyczną *OpenFile(std::string path, std::string mode)*. Metoda *OpenFile* zwraca wskaźnik na *IFile*. Wskaźnik ten wskazuje na dynamicznie zaalokowany obiekt odpowiedniej klasy pliku, w zależności od podanego rozszerzenie pliku.
e. Przetestować działanie klasy *FileFactory* otwierając pliki różnego typu.

4. Sprawozdanie

Stworzone oprogramowanie, implementujące fabrykę obiektów, zgodnie z przyjętym wzorcem. Interfejs *IFile*, po którym dziedziczyć muszą klasy *CsvFile* oraz *BinaryFile*, które muszą umożliwiać działanie na plikach, według laboratorium 2 i 3.

Pliki:

- IFile.h
- IFile.cpp
- FileFactory.h
- FileFactory.cpp
- CsvFile.h
- CsvFile.cpp
- BinaryFile.h
- BinaryFile.cpp
- main.cpp - z testami działania FileFactory