Dokumentacja projektu

Projekt numer 6: Przetwarzanie własnych typów danych
CLR UDT

Nazwa projektu: SchoolUtils

autor: Długosz Bartłomiej
numer indeksu: 297856
uczelnia: Akademia Górniczo-Hutnicza
wydział: Fizyki i Informatyki Stosowanej
rok: 3.

---

1. Opis problemu.

Celem projektu było stworzenie aplikacji konsolowej wykorzystującej złożone **UDT** (**User Defined Types)**. Ma ona zawierać niezbędne testy jednostkowe, a także udostępniać opcję dodawania rekordów, wyszukiwania oraz tworzenia raportów.

Jako typy złożone zostały wybrane przybory szkolne – kalkulator, długopis, zeszyt, klej, gumka do mazania, linijka oraz nożyczki.

2. Opis funkcjonalności udostępnianej przez API.

Stworzona aplikacja jest aplikacją konsolową – użytkownik odpowiednio porusza się po niej dzięki odpowiedniej funkcjonalności, a także instrukcjach widniejących bezpośrednio w konsoli.

Poruszanie się po aplikacji jest intuicyjne dzięki wyboru odpowiednich cyfr oznaczających przejście do danego okna.

Użytkownik ma możliwość dodawania oraz przeszukiwania odpowiednich przyborów szkolnych oraz generowania odpowiednich raportów z nimi związanych.

2.1. Menu główne.

Menu główne prezentuje się następująco:

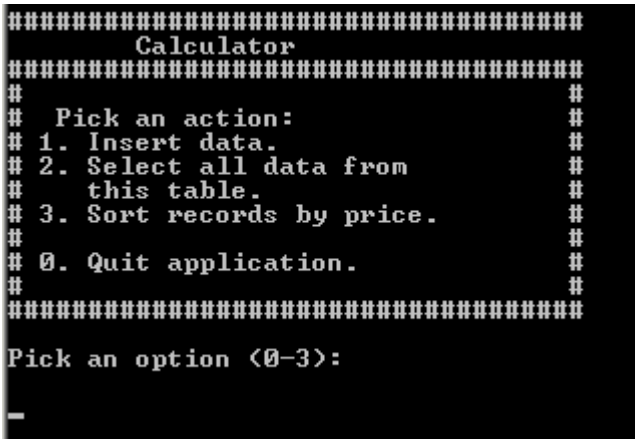
gdzie wylistowane nazwy oznaczają okna **UDT**, w którym możemy dokonywać wszelkich operacji:

- **1. Calculator** – oznacza złożony typ UDT – kalkulator,
- **2. Glue** – oznacza złożony typ UDT – klej,
- **3**. **Notebook** – oznacza złożony typ UDT – zeszyt,
- **4. Pen** – oznacza złożony typ UDT – długopis,
- **5. Rubber** – oznacza złożony typ UDT – gumka do mazania,
- **6. Ruler** – oznacza złożony typ UDT – linijka,
- **7. Scissors** – oznacza złożony typ UDT – nożyczki,
- **0. Quit application –** oznacza wyjście z aplikacji .

Aby wybrać daną opcję, należy wpisać odpowiednią cyfrę, a następnie zatwierdzić instrukcję klawiszem **Enter**.

2.1. Menu typów UDT.

Menu typów UDT prezentuje się nastepująco:



gdzie wylistowane nazwy oznaczają odpowiednie operacje na bazie danych:

- **1. insert data** – umożliwia dodania rekordu do danej tabeli, zostaje wyświetlona odpowiednia instrukcja pouczająca jak dany rekord powinien wyglądać,
- **2. select all data** – wylistowuje wszystkie rekordy tej tabeli znajdujące się w bazie danych,
- **3**. **sort records by price** – wylistowuje wszystkie rekordy tej tabeli znajdujące się w bazie danych, posortowane według ceny (od najmniejszej do największej)
- **0. quit application –** oznacza wyjście z aplikacji.

Po wybraniu odpowiedniej operacji zostanie wyświetlony komunikat lub rezultat zapytania oraz informacja o powodzeniu dokonania danej instrukcji. Po wciśnięciu dowolnego przycisku następuje powrót do menu głównego.

3. Opis typów danych oraz metod (funkcji) udostępnionych w ramach API, szczegóły implementacji.

Aplikacja składa się z klas: **Command,** głównej klasy **Program** oraz siedmiu klas reprezentujących typy złożone: **Calculator, Pen, Notebook, Glue, Rubber, Ruler** oraz **Scissors.**

3.1. Klasa Command.

Klasa ta reprezentuje komendę SQLową.

Pola klasy:

a) ***m_Comm -*** przechowuje treść zapytania lub jego część, która jest stała w każdym wywołaniu,
b) ***m_Type -*** przechowuje informację o typie zapytania – jest to enum, w którym są następujące pola: **Null**, **Insert**, **SelectAll**, **SortByPrice**,
c) ***m_Prompt -*** wiadomość wyświetlana przy odpowiednich operacjach na bazie danych,
d) ***m_ResultList*** - zawiera listę nazw kolumn, które będą zwrócone po zapytaniu.

Metody klasy:

- **public String GetCommand()** - getter na wartość **m_Comm** klasy,
- **public Type GetCommandType()** - getter na wartość ***m_Type*** klasy,
- **public String GetPrompt()** - getter na wartość ***m_Prompt*** klasy,
- **public List<string> GetResultList()**- getter na wartość ***m_ResultList*** klasy

3.1. Klasa Program.

Pola klasy:

a) **Dictionary<string, Command> _commands –** słownik (mapa) zawierająca wszystkie możliwe komendy SQLowe**,**
b) **Dictionary<int, string> _names -** słownik (mapa) zawierająca nazwy klas UDT (pomocniczy słownik do koordynacją z menu głównym programu)**,**
c) **Dictionary<int, string> _prescriptions -** słownik (mapa) zawierająca opisy odpowiednich instrukcji, które pojawiają się na ekranie konsoli podczas użytkowania aplikacji.

Metody klasy:

- **public static void PrintMenu() -** wyświetla główne menu na konsolę,
- **private static void Setup() -** przygotowuje pola **_commands, _names, _prescriptions** tuż przy rozpoczęciu programu,
- **static void SendCommand(Command c, int num) –** przyjmuje referencję do klasy **Command**, a także wartość **int,** która identyfikuje wybrany typ **UDT**. Funkcja ta nawiązuje połączenie z bazą danych, wysyła odpowiednie zapytanie SQLowe oraz obsługuje odpowiedź, a także ewentualne wyrzucone wyjątki,

- **public static void PrintUtilMenu(int num) -** wyświetla menu edycji bazy danych konkretnego **UDT** na konsolę, umożliwia interakcję z bazą danych a także na wykonywanie odpowiednich zapytań SQLowych**,**
- **static void Main(string[] args) –** główna funkcja programu

3.2. Klasy Calculator, Pen, Notebook, Glue, Rubber, Ruler oraz Scissors.

Klasy te reprezentują typy złożone, jakimi są przybory szkolne. Implementują one interfejsy: **Inullable** oraz **IbinarySerialize**.

Pola klasy:
Każda klasa zawiera następujące pola:

- **double m_Price –** cena produktu,
- **private bool m_Null –** flaga, czy dana referencja jest niezainicjalizowana.

Metody klasy:
Każda klasa zawiera następujące metody:

- **public string GetName()** - zwraca nazwę klasy,
- **public double GetPrice()** - zwraca cenę danego produktu,
- **public string ToString()** - zwraca w postaci stringa reprezentację danego obiektu wraz z opisem jego wartości,
- **public static <nazwa_klasy> Parse(SqlString s) –** zwraca nowo utworzony obiekt z polecenia podanego jako argument wywołania metody,
- **private bool Validate()** - metoda, która waliduje poprawność wprowadzonych danych,
- **public void Write(System.IO.BinaryWriter w) –** metoda serializująca obiekt,
- **public void Read(System.IO.BinaryReader r) –** metoda deserializująca obiekt.
- **public static <nazwa_klasy> Null** - metoda zwracająca nowy obiekt, którego referencja jest nullem.

Oprócz powyższych podstawowych pól i metod klasy te posiadają własne dodatkowe pola wraz z odpowiednimi getterami na te pola. Ich opis oraz działanie jest trywialne.

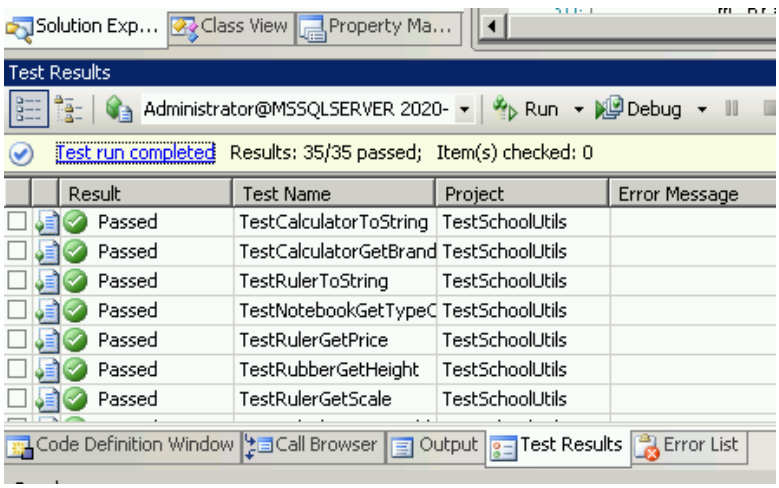
5. Testy jednostkowe

Zostały przygotowane testy jednostkowe, które testują odpowiednie metody wszystkich klas, a także testy SQLowe, które widnieją w pliku **all_tests.sql**.

W pliku **all_tests.sql** zostały przygotowane odpowiednie zapytania, zapytania wykomentowane znakiem '--' są niepoprawne.

Łącznie wykonano 35 testów jednostkowych dla metod klas, zaimplementowanych w języku C#, sprawdzających poprawność wyników, które zwracają metody.

Wszystkie testy klas znajdują się w podprojekcie TestSchoolUtils, pliki te rozpoczynają się od przedrostka **UnitTest.**
Wszystkie testy klas znajdują się w pliku **all_tests.sql**.



6. Kod źródłowy.

Cały kod źródłowy znajduje się w katalogu z projektem.

7. Podsumowanie i wnioski.

Aplikacja została napisana w języku C#, umożliwia użytkownikowi operowanie na złożonych typach UDT poprzez oferowane API, dzięki któremu wchodzi on w interakcję z programem oraz bazą danych.

Opracowane API umożliwia wprowadzanie danych do bazy, wyszukiwanie rekordów a także generowanie raportów.

W projekcie zostały zastosowane testy jednostkowe, które sprawdzają poprawność implementacji wszystkich metod.

8. Bibliografia.

- https://newton.fis.agh.edu.pl/~antek/read_pdf.php?file=BD2_L09_CLR.pdf,
- https://mmazurek.dev/udt-czyli-wlasne-typy-danych-w-mssqlu/?fbclid=IwAR0-ms1fV2RGyB9tF5aPfPcl8lDp39XBYA37vFEjd8gsS3JFSBspckuEFRg,
- https://stackoverflow.com/,
- https://docs.microsoft.com/en-us/