CODE SNIPPET MANAGER

Aplikacja zaprojektowana i zaprogramowana przez Jana Solicha

Opis

SnippetManager („menedżer kawałków kodu”) to program napisany w C# i zaprojektowany do zbierania fragmentów kodu (ang. „snippet”) z różnych języków programowania do celów informacyjnych/nauki różnych funkcjonalności wybranego języka. Na przykład, dla C# byłoby kilka fragmentów pokazujących, jak używać delegatów lub pisać metody rozszerzeniowe, dla Javy przykłady użycia strumieni („streams”), dla C++ przykłady użycia klasy bitset, a dla Pythona przykłady użycia funkcji „list comprehension”.

Funkcjonalności

* Graficzny interfejs użytkownika
* Dodawanie, edytowanie, usuwanie fragmentów kodu
* Klonowanie fragmentów kodu w celu testowania ich różnych wariacji
* Podświetlanie składni kodu dla wszystkich wspieranych języków
* Możliwość zapisania danych kawałków kodu w pliku i odczytania ich z tego pliku w dwóch trybach: zastępowania (wszystkie snippety z listy zostaną zastąpione tymi z pliku) i dodawania (snippety z pliku zostaną dodane do istniejących, a istniejące zostaną).
* Wsparcie dla wielu języków programowania: Lua, Python, C#, C++, Java
* Kategoryzowanie fragmentów według tematu, języka programowania, poziomu zaawansowania
* Niektóre fragmenty mogą być uruchamiane bezpośrednio w aplikacji, aby natychmiast sprawdzić wynik i zachowanie, szczególnie po zmodyfikowaniu fragmentu (wspierane języki: Lua, Python, C#).
* W językach dynamicznych (Lua, Python) istnieje możliwość zachowania stanu interpretera między kolejnymi uruchamianymi snippetami. To znaczy np., że zmienne globalne zdefiniowane w snippecie będą dostępne i będą miały taką samą wartość w każdym innym snippecie w tym języku (oczywiście dopóki nie zostaną zmienione).
* Rozszerzony opis dla niektórych fragmentów (wraz z linkami do stron, gdzie omawiany jest temat), dostępny po kliknięciu przycisku

Różne uwagi

* Cały projekt będzie podzielony na bibliotekę implementującą większość najbardziej rozbudowanych funkcjonalności oraz aplikację GUI korzystającą z tej biblioteki.
* Interfejs GUI powinien mieć górne menu, jak w większości Windowsowych programów, i używać ikon, np. na przyciskach lub ikony pliku wykonywalnego programu (jeszcze niezaimplementowane).
* Program powinien używać jakiejś bazy danych do przechowywania fragmentów kodu (może to być po prostu JSON/XML).
* Należy zrobić przeglądarkę/edytor fragmentów kodu wspierającą języki programowania, w których może być potrzebnych wiele plików dla pojedynczego fragmentu (C++).

Implementacja

Biblioteka główna dostarcza klasy „CodeSnippet” i „SnippetList” reprezentujące główne elementy aplikacji. Ta pierwsza zawiera methody np. do uruchamiania kawałków kodu w wspieranych językach. Druga zawiera methody np. do filtrowania listy kawałków kodu według różnych kryteriów i zapisywania/odczytywania z pliku. Serializacja jest zrealizowana za pomocą wbudowanej klasy JsonSerializer.

Aplikacja GUI (WinForms) korzystająca z biblioteki zawiera dwie klasy okien („formularzy”). Jedno główne umożliwia użycie większości funkcjonalności (filtrowanie snippetów, prosta edycja [tylko kod snippetu], uruchamianie, usuwanie, klonowanie, odczyt i zapis do pliku…). Drugie pomocnicze umożliwia dodawanie/dokładną edycję kawałków kodu, umożliwiając dostosowanie praktycznie każdej właściwości snippetu.

Zawarta jest też klasa rozszerzająca klasę Scintilla z pakietu NuGet Scintilla5.NET. Scintilla to mocno zaawansowany edytor kodu napisany w C i używający Windowsowych komunikatów do zarządzania; Scintilla5.NET to wrapper .NET do tego edytora, umożliwiający korzystanie ze znacznej większości funkcjonalności w łatwy sposób i uwalniający od niskopoziomowych detali jak np. sterowanie komunikatami Windows.

Moja klasa dziedzicząca po klasie z pakietu dostarcza kilka przydatnych funkcjonalności, np. dynamiczna zmiana języka (w tym wypadku trzeba zrobić kilka rzeczy, które są „opakowane” przez jedną funkcję).

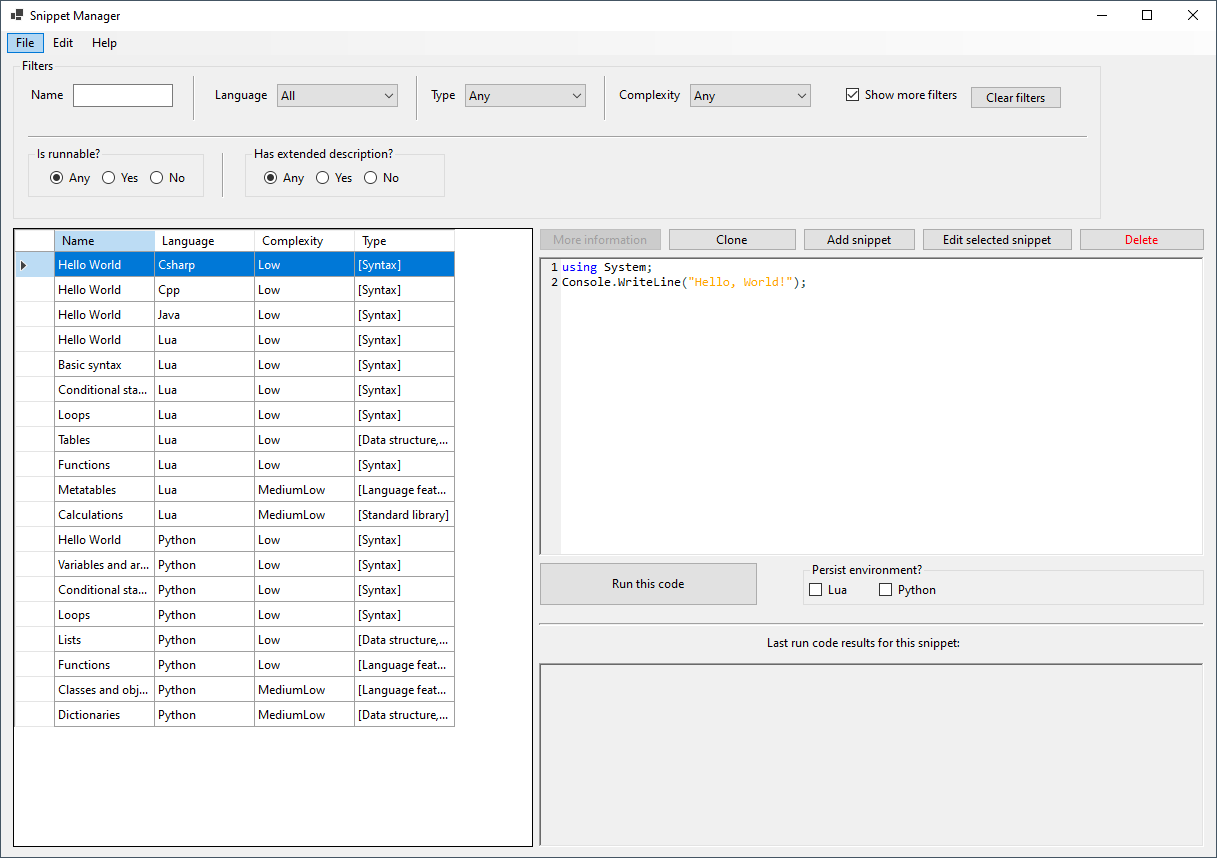
Kod programu

Do przechowywania kodu używam systemu kontroli wersji „Git” na platformie GitHub. Link do repozytorium: <https://github.com/Eksekk/SnippetManager>

Działająca wersja programu do natychmiastowego uruchomienia jest załączona w pliku .zip.

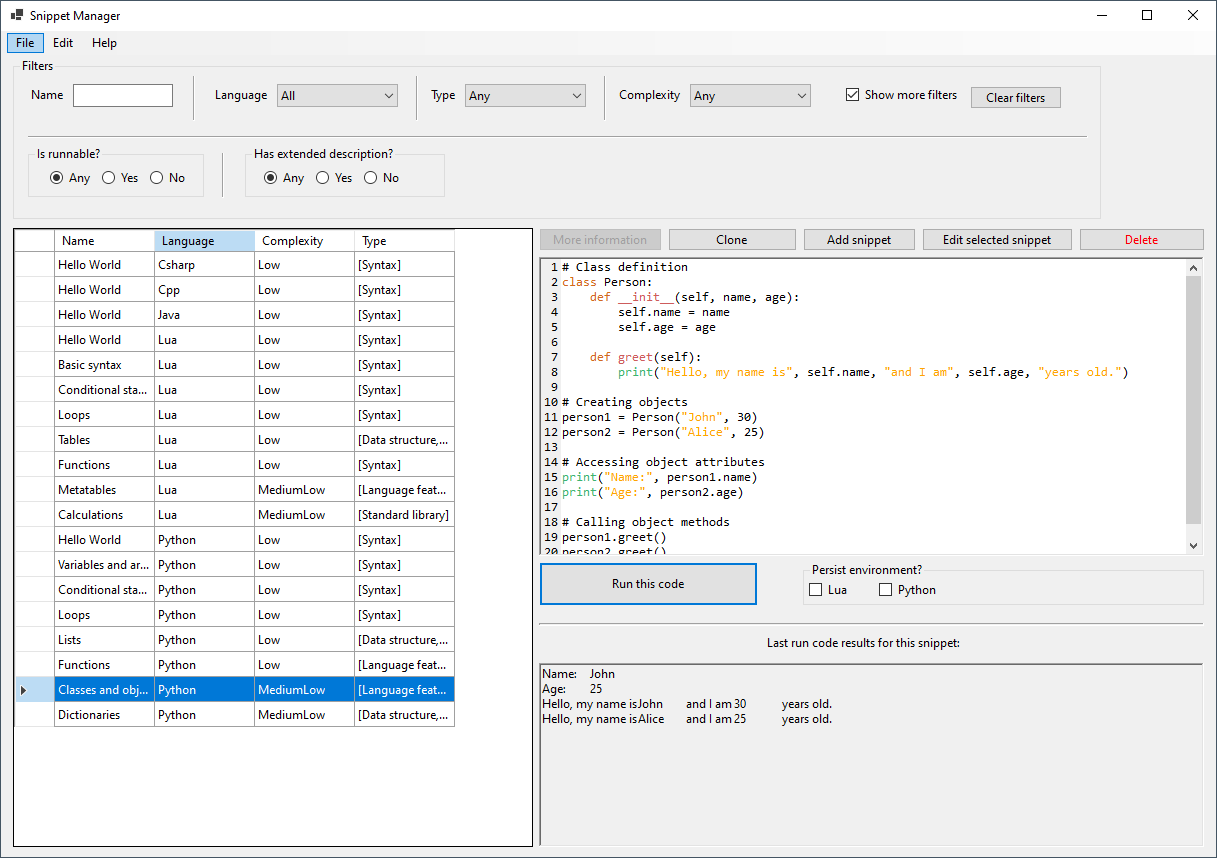
Zrzuty ekranu

Poniżej znajduje się kilka przykładowych zrzutów ekranu prezentujących aplikację.



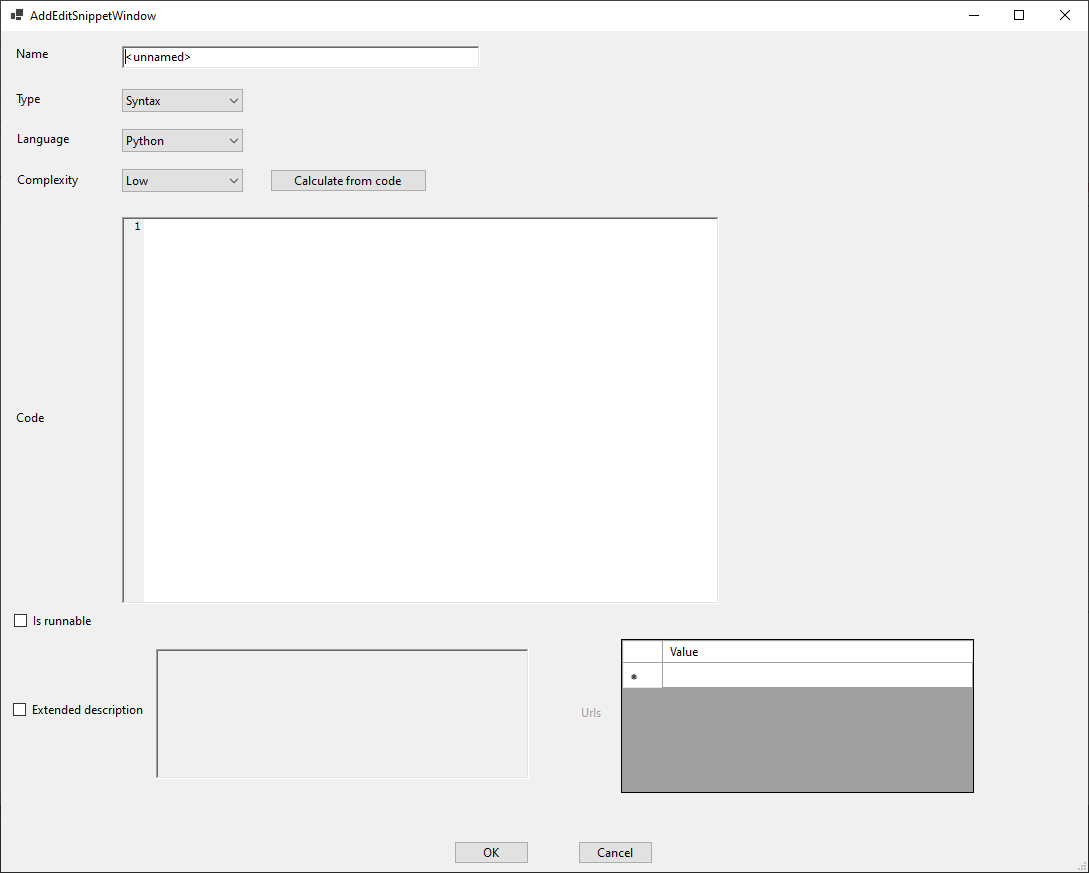
Główne okno aplikacji

Na obrazku wyżej widzimy główne okno aplikacji. Menu aplikacji pozwala m.in. zapis i odczyt z pliku. U góry są filtry pozwalające pokazywać snippety spełniające określony warunek. Po lewej jest lista snippetów. Po prawej jest okno edytora kodu razem z przyciskami pozwalającymi wykonać zaawansowane operacje na snippetach. U dołu po prawej jest konsola pokazująca wyniki uruchomienia kodu.



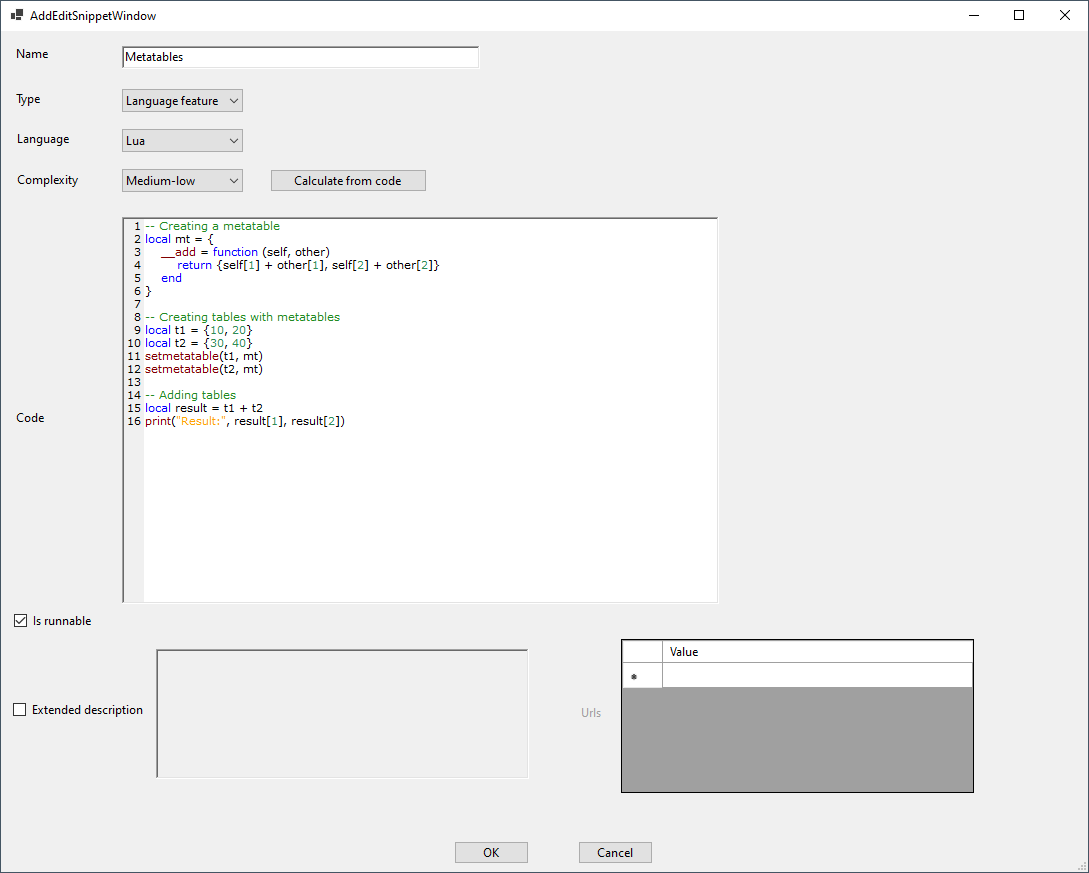
Okno aplikacji po wykonaniu kodu snippetu

Powyżej widać efekt kliknięcia przycisku „run this code”. Wszystkie wywołania funkcji pythona „print” są przechwytywane i wypisywane na konsolę.



Okno dodawania snippetu

Powyżej znajduje się okno dodawania snippetu. Wszystkie pola mają domyślną wartość albo są puste. Wyjaśnienie funkcjonalności nowych elementów interfejsu będzie poniżej.



Okno edytowania snippetu

Powyżej przedstawione jest ta sama formatka, ale otwarte w trybie edycji. Obie funkcjonalności używają tego samego okna, przy czym przy otwieraniu do edytowania, elementy formatki są ustawiane na wartości edytowanego snippetu. Nowe elementy:

* Przycisk „Calculate from code” – automatycznie liczy stopień skomplikowania kodu i go ustawia. Użyty algorytm to tylko przykład, jest bardzo prymitywny.
* Checkbox „Is runnable” – ustawia, czy dany snippet jest oznaczony jako możliwy do uruchomienia.
* Elementy z „Extended description” – pierwsze okienko pozwala wpisać dłuższy opis snippeta, a drugie pozwala na wygodnie dodawanie adresów URL do wyświetlenia.

Użyte paczki (packages) z NuGet

Projekt używa kilku paczek stworzonych przez innych, w celu skrócenia czasu tworzenia programu i dostarczenia funkcjonalności, których nie mogę sam zaprogramować z jakiegokolwiek powodu (czas, brak umiejętności itp.). Oto one:

* *IronPython* ([ironpython.net](https://ironpython.net/)) – umożliwia uruchamianie kodu napisanego w Pythonie. Jest to Microsoftowa implementacja języka Python, ściśle zintegrowana z platformą .NET.
* *NeoLua* ([*github.com/neolithos/neolua*](https://github.com/neolithos/neolua/))– umożliwia uruchamianie kodu napisanego w Lua. Jest to implementacja Lua napisana przez programistę „Neolithos”. Zauważalnie różni się od „zwykłej” implementacji Lua, ale za to jest ściśle zintegrowana z platformą .NET.
* *Scintilla5.NET* ([github.com/desjarlais/Scintilla.NET](https://github.com/desjarlais/Scintilla.NET)) – udostępnia zaawansowany edytor kodu Scintilla. Jest to wrapper nad oryginalną Scintillą, która jest napisana w C i używa windowsowych „message” do interakcji z każdym innym kodem. Dzięki wrapperowi, nie mamy styczności z niskopoziomowymi detalami i możemy korzystać z bogatego i respektującego konwencje .NET api.
* *Lib.Harmony* ([github.com/pardeike/Harmony](https://github.com/pardeike/Harmony)) – jest to biblioteka, która umożliwia łatwe zmienianie kodu programu **podczas jego wykonywania**. Możemy np. nadpisać funkcję, wykonać dowolny kod przed wywołaniem funkcji, zmienić dowolną instrukcję CIL itp. Używam jej do nadpisania metody Console.WriteLine tak, żebym mógł przechwycić wiadomość i wyświetlić ją na konsoli.