Popis návrhu rozhraní

Základní myšlenkou návrhu rozhraní je odprostění se od konkrétní implementace. Naše API například umožňuje jednoduše implementovat rozdílné zdroje a cíle uložení dat. Nebude proto problém ini soubory nahradit databází a načtenou konfiguraci pak uložit třeba ve formátu XML. Při návrhu rozhraní jsme dále měli na mysli loose coupling a single responsibility princip, kde se funkcionalita dá rozdrobit do menších modulů s jasně vymezenou odpovědností. Tento návrh bude podle nás podporovat i unit testování, protože budeme schopni potřebné moduly jednoduše mockovat.

Popis základních částí knihovny

IConfig

Základním centrálním kamenem je `IConfig`. `IConfig` má jediný účel - reprezentovat data v době, kdy už jsou načtena v paměti a inicializována, ale předtím než jsou zase uložena. Tedy přesně tehdy, když je programátor chce používat uvnitř svého programu. Ostatní objekty už nějak tato data použijí, to ale není odpovědnost `IConfig`. `IConfig` pak pouze zprostředkovává přístup k sekcím konfiguračního souboru a jejich správu.

IConfigSection

Interface `IConfigSection` reprezentuje konkrétní sekci konfiguračního souboru a obsahuje správu jednotlivých konfiguračních voleb.

IConfigBuilder

Interface `IConfigBuilder` umožňuje implementaci načítače konfigurace z nějakého datového zdroje. Pro naše účely bude pravděpodobně implementován konkrétní `IniConfigBuilder`, v budoucnu může být ale nahrazen například třidou `XMLConfigBuilder` a dalšími. Zároveň tak bude možné implementovat `MemoryConfigBuilder` pro testovací účely.

IConfigSaver

Podobně jako u `IConfigBuilder` umožňuje tento interface implementaci různých cílů uložení konfiguračních dat. Jak už to udělá závisí na implementaci, pouze na vstupu obdrží `IConfig` objekt. V našem scenáriu opět použijeme nějakou variantu `IniConfigSaver` pro práci se soubory a `MemoryConfigSaver` pro testování.

IFormatSpecifier

`IFormatSpecifier` bude implementován objektem, pomocí kterého uživatel definuje, jak má vypadat načítaný konfigurační soubor. Objekt tedy zapouzdří práci s pravidly, která budou použita při načítání konfiguračního souboru.

ConfigValue

Seznam typů, které lze v configu reprezentovat je vymezen implementacemi potomků třídy `ConfigValue`. ConfigValue je místo, kde budou uložena skutečná data každé volby.

Typy

Typy, které je možné v konfiguračním souboru ukládat, jsme obalili do vlastních objektů (potomci třídy **ConfigValue**), které umožňují parsovat hodnoty. Tyto objekty jsou poté použity i ve specifikaci formátu konfiguračního souboru.

Výčtové typy a seznamy jsme vyřešili vlastní speciální třídou. Oboje opět dědí od ConfigValue, čímž jsme omezili, jaké typy se v nich mohou nacházet.

Omezování hodnot položek konfiguračního souboru (například vymezení intervalu pro čísla) jsme vyřešili pomocí dosazení libovolného predikátu, kterým uživatel naspecifikuje libovolnou podmínku. Lze tímto teoreticky také vyřešit i výčtové typy.

Atributy

Kromě standardního použití config knihovny, kde uživatel ručně popíše konfigurační soubory a procedurálně je zpracovává, jsme vytvořili i druhý způsob, který je založen na anotacích tříd pomocí atributů. Programátor má možnost definovat třídu, které reprezentuje datový model a potom přímo na field memebery této třídy navázat hodnoty z konfiguračního souboru. Knihovna pak pomocí reflekce najde tyto anotace a hodnoty objektu naplní. Odpadává tím tak nutnost ručně načítat hodnoty ze souboru a ty pak ukládat do modelu.

Ukázku použití atributů lze naleznout v projektu **Examples** v souboru **AttributeUsage.cs**.

Ukázka použití

Jak lze knihovnu použít je ukázáno v projektu **Examples** hlavního solution. V souboru **BasicUsage.cs** jsou uvedeny příklady klasické práce s konfiguračním souborem od jeho načtení až po jeho uložení. Je zde použito několik různých objektů od Builderu až po Saver. Ve finální implementaci ovšem nevylučujeme, že všechny tyto interfacy zastřeší jediný objekt, který bude obsahovat veškerou funkcionalitu - například nějaký **IniConfig**.