Dokumentation des „Person Data Management System“

Beschreibung der Funktionalität:  
Das „Person Data Management System“ ist ein Programm zum Verwalten von Personen. Nutzer können Personen anlegen, löschen, filtern, exportieren und importieren.

* Anlegen: Beim Anlegen einer Person wir deine neue Sicht (Fenster) geöffnet in dem der Nutzer eine neue Person mit dessen Eigenschaften definieren kann. Die Sicht ermöglicht dem Nutzer die Auswahl eines Personentyps und gibt die entsprechenden Eigenschaften der ausgewählten Person zur Bearbeitung frei. Beim Wechseln des Personentyps werden ausgefüllte Felder die beide Personentypen beherbergen mit ihren Werten beibehalten. Beim Betätigen des „Save“ Knopfes wird die bearbeitete Person im Kontext der Anwendung gespeichert und steht in der Übersicht bereit. Beim Betätigen des „Cancel“ Knopfes werden alle Änderungen verworfen.
* Löschen: Beim löschen eines Nutzers (Klick auf den Löschen Knopf hinter dem Personen Eintrag) wird die Person aus dem Anwendungskontext gelöscht und zugehörige Element aus der Sicht entfernt.
* Filter: Beim betätigen des „Manage Filters“ Knopfes öffnet sich eine neue Sicht (Fenster) in der der Nutzer Filter für alle vorhandene Personen setzen. Zudem existiert standartmäßig ein unlöschbarer Filter für den Personentyp. Mit dem „Add Filter“ Knopf wird ein neuer leerer Filtereintrag erzeugt, der aus 3 Teilen besteht. Der zu filternden Eigenschaft, dem Vergleichsoperator (>, <, =, Contains) und dem Wert, dem es zu vergleich gilt. Um einen Vergleichsoperator festlegen zu können muss erst eine Eigenschaft ausgewählt werden, da der Operator abhängig von der Eigenschaft ist. Um den Wert des Filters festzulegen muss erst festgelegt werden welcher Vergleichsoperator verwendet wird. Wird die zu filternde Eigenschaft geändert werden die folglich davon abhängigen Felder auf den initialwert zurückgesetzt(Vergleichsoperator und Wert Feld). Der „Delete“ Knopf hinter dem Filter Eintrag entfernt den Filter aus der Anwendung, der „Save“ Knopf übernimmt alle eingestellten Filter in den Anwendungskontext. Alle in der Anwendung gespeicherten Personen werden nun mit den gesetzten Filtern überprüft und beim Erfolgreichen durchlaufen aller Filter angezeigt. Beim Bestätigen des „Cancel“ Knopfes werden die Filtereinstellungen nicht in den Anwendungskontext übernommen und dementsprechend verworfen.
* Export: Beim betätigen des „Export Persons“ Knopfes werden alle in dem Anwendungskontext gespeicherten Personen Serialisiert und in eine JSON Datei geschrieben.
* Import: Beim betätigen des „Import Persons“ Knopfes werden alle Daten aus einer JSON Datei deserialisiert und in den Anwendungskontext geladen.

## Anmerkung:

Die XAML Dateien nutzen Grids und skalieren dementsprechen dynamisch mit. Dies wird aber im Folgenden nicht erwähnt, da es viel Text für ein wieder auftretendes Element wäre.

## Komponentenbeschreibung:

**MainWindow**: Hauptfenster der Anwendung. Beinhaltet das UserControl Element „PersonOverview“ und den Titel der Anwendung.

**PersonOverview:** Ist ein „UserControl“ Element welches in der dazugehörigen XAML Datei die Knöpfe zum Hinzufügen, exportieren, importieren und verwalten von Filtern beschreibt. Die XAML Datei beschreibt außerdem die Auflistung der „PersonEntry“ UserControl Elemente anhand eines WrapPanels. Die PersonOverview registriert im constructor die updatePersonTable Methode auf die Events „personAdded“, „personSet“, „personRemoved“ und “filtersClosed”. Beim Ändern der PersonCollection oder beim Schließen des Filter Fensters wird dadurch die Methode ausgelöst. Die Methode updatePersonTable löscht dabei alle Kind elemente des WrapPanels und bestückt diese erneut mit jedem Element der PersonCollection, die in dem AppState (AnwendungsKontext) hinterlegt ist. Beim Bestücken wird durch verwenden der filterCanBeApplied Methode geprüft ob die Filter auf die Elemente (Personen der PersonCollection) angewandt werden können. Ist dies nicht der Fall, wird die Person nicht in die View bzw. das WrapPanel geladen. Die filtersCanBeApplied Methode prüft dabei ob die Person die im Filter festgelegte Eigenschaft besitzt, anschließend wird der anhand der „Comparator“ Eigenschaft des Filters festgelegt wie der vergleich stattfinden soll und dann wird der Eigenschaftswert der Person mit dem des Filters verglichen. Nach durchlaufen jeder Prüfung wird entweder ein true für bestehen oder false für nicht bestehen des Filters in eine Liste von bool Werten (filtersCouldBeApplied) hinzugefügt. Wenn diese Liste am Ende der Methode nur wahre Werte (d.h. alle Filter konnten erfolgreich durchlaufen werden) beinhaltet wird ein true zurückgegeben, andernfalls ein false. Ein Sonderfall in dieser Funktion ist ein selbstdefinierter Filter (nicht vom Typ Filter), welcher den ausgewählten Datentyp überprüft.  
Die Funktionen import- und exportUsers\_Click serialisieren bzw. deserialisieren die Personen für den AppState (Anwendungskontext) mittels des vom NuGet bezogenen „Newtonsoft.Json“ Serialisierers. Die NuGet erleichert das Serialisieren und Deserialisieren unter Beachtung der instanziierten Typen. (Wichtig um beim laden nicht die Instanztypen zu verlieren). Das Speichern und Laden der Serialisierten Daten erfolgt dabei durch die „File“ Klasse. Die Methode mangeFilter\_Click erstellt lediglich eine neue „FilterOverview“ und öffnet diese. Die „removeEntry“ Methode entfernt ein element aus dem WrapPanel. Die Methode wird ausschließlich von den PersonEntries verwendet.

**PersonEntry:** Die „PersonEntry“ XAML besteht Haupstächlich aus drei Elementen: PersonName, einem Knopf zum Auslösen der viewUser Methode und einem Knopf zum Auslösen der deleteButton\_Click Methode. Der Konstruktor der Klasse initialisiert die Texte der UI Elemente und speichert sich die zu verwaltende Person und den Parent zum Löschen der „PersonEntry“.

**PersonView:** Die „PersonView“ XAML besitzt eine Überschrift, eine ComboBox zum auswählen des Personentyps, ein WrapPanel in dem die Eigenschaften des entsprechenden Typs dynamisch angezeigt werden und zweit Knöpfe: save und cancel. Der Konstruktor nimmt eine Personeninstanz entgegen (eine neue Person ist dabei einfach nur eine Neue Instanz der Personen Klasse) und speichert sich diese als Backup. Zudem wird der Index des Objekts ermittelt um ggf. später das Element überschreiben zu können. Die InitializeCombBox Methode, welche im Konstruktor aufgerufen wird, ermittelt alle Werte aus dem PersonModelDictionary, konvertiert diese zum Simplen Klassennamen (Klasse ohne Pfad) und fügt diese der ComboBox als Items hinzu. Zudem wird ein EventHandler(onUpdateUserControl) dem event DropDownClosed hinzugefügt, sodass die PropertyEntries bei Auswahl eines Elements neu geladen werden. Die UpdateUserControll Methode löscht alle Elemente des WrapPanels und bestückt diese anschließend neu. Dabei wird als erstes eine neue Instanz des ausgewählten Typs erstellt und der PersonToEdit zugewiesen. Diese dient als Referenz für die Reflection um an die Ziel Eigenschaften zu kommen. Anschließend werden die Funktionen updateUserControlByCommonClassFields und updateUserControlByUniqueClassFields aufgerufen, um alle gemeinsamen und nicht gemeinsamen Eigenschaften (des aktuellen Typs/Models und des Ziel Typs) zu ermitteln und neu zu erstellen. Gemeinsame Eigenschaften werden Dabei in der findCommonFields Methode als sharedProperties beibehalten. Die save\_Click Methode fügt der PersonCollection des AppState’s (Anwendungskontextes) im Falle eines neuen Elements die Aktuelle Personen Instanz hinzu oder überschreibt sie, wenn sie bereits existiert. Beim Schließen des Fensters durch den cancel Knopf die cancel\_click methode wird das backup geladen und das Fenster geschlossen.

**PropertyEntry:** Die „PropertyEntry“ XAML besteht aus einem Label (Name der Eigenschaft) und einer Textbox (Wert der Eigenschaft). Der Konstruktor nimmt eine Person und einen Namen (Namen der zu bindenden Eigenschaft) entgegen und erstellt ein TwoWay Databinding zur entsprechenden Eigenschaft.

**FilterOverview:** Die „FilterOverview“ XAML besitzt ein WrapPanel um FilterEnties anzuzeigen (ähnlich wie bei der PersonOverview und dessen PersonEntries) und zudem ein Stackpanel für einen gesonderten, nicht löschbaren Filter (nicht vom Typ Filter, dies ist ein Sonderfall, da es keine Eigenschaft eines Datentyps ist) der zum Filtern des Personentyps zuständig ist. Dieser besteht aus einem Label und einer Combobox (ebenfalls mit Werten aus dem PersonModelDictionary initialisiert) dessen Standard Wert „Person“ ist um alle Elemente anzeigen zu können. Der Konstruktor initialisiert dabei wie gerade beschrieben den PersonenTyp Filter und dabei noch alle im AppState definierten Filter. Dies geschieht ähnlich wie bei der „PersonOverview“, nur dass hier ein FilterEntry statt eines PersonEntry verwendet wird. Beim betätigen des AddFilter Knopfes wird der eventHandler AddFilter\_Click ausgelöst und erstellt einen neuen Filter mit dem ein neuer FilterEntry erzeugt wird. Der save Knopf löst den eventHandler saveButton\_Click aus. Dieser löscht alle existierenden Filter und bezieht sich die neuen Filter aus der aktuellen View bzw. aus den Kind Elementen des WrapPanels. Anschließend das Fenster geschlossen und ein event ausgelöst (filtersClosed) um die „PersonOverview“ aktuell zu halten. So wie die PersonOverview auch besitzt die FilterOverview ebenfalls eine Methode zum löschen einer FilterEntry (wird ebenfalls nur vom entry selber verwendet).

**FilterEntry:** Die „FilterEntry“ XAML besteht aus einer ComboBox zum auswählen der zu filternden Eigenschaft, einer ComboBox zum auswählen des Vergleichsoperators, einer Textbox zum eingeben des zu vergleichenden Wertes und einem Kopf zum Löschen des Eintrages. Der Konstruktor erhält ein Filter Element, dass es gilt zu binden und das parent Element zum Löschen des Eintrages. Der Konstruktor erstellt dabei Datenbindungen zu alle Feldern des Filters und weißt die eventHandler (onPropertyNameComboBoxChanged und onComparatorBoxChanged) dem event SelectionChanged der korrespondierenden ComboBoxen zu. Diese Handler sichern durch Neu Initialisierung der UI Elemente ab, dass keine ungültigen Eingaben stattfinden können. Die InitializeComparatorValues Methode bestückt je nach ausgewähltem Eigenschaftstyp (double, int32 oder string) die ComboBox die den Vergleichsoperator angibt. Wie die die „PersonEntry“ auch besitzt die Klasse auch einen eventHandler (RemoveButton\_Click) zum Entfernen der FilterEntry mittels des Parents.

**PersonCollection:** Die „PersonCollection“ beinhaltet die events personAdded, personSet und personRemoved, die am Ende der Methoden InsertItem, SetItem und RemoveItem ausgelöst werden. Die genannten Methoden sind von der Collection<Person> vererbt und überschrieben. Sie besitzen ein eigenes Fehlerhandling.

**PersonModelDictionary:** Das „PersonModelDictionary“ enthält ein statisches readonly dictionary, welches alle Personen Models, die der Nutzer erstellen können und filtern können soll, referenziert. Bestehend aus dem Key (simpler Klassenname) und der Value (volle Typbezeichnung + namespace).  
Die Klasse enthält zwei Funktionen:

* GetValue(): liefert einen gültigen Wert zurück. Loggt bei ungültigem Wert eine Exception in der Konsole.
* GetSimpleClassFromType(string value): erzeugt den simplen Namen einer Klasse oder um es anders zu beschreiben den Key.

**ClassGenerator:** Der „ClassGenerator“ hat eine statische Methode um aus den Werten des PersonModelDictionary eine Instanz des jeweilig referenzierten Models zu erstellen.

**AppState:** Enthält den Anwendungskontext bestehend aus drei Eigenschaften:

* PersonCollection: Eigene Collection die alle Personen enthält.
* Filters: Alle anzuwendenden Filter basierend auf Eigenschaften.
* PersonTypeFilter: Sonder Filter, der nicht auf einer Modeleigenschaft basiert.

**Filter:** Enthält drei Eigenschaften:

* PropertyName: Name der zu filternden Eigenschaft
* Comparator: Vergleichsoperator
* ValueToCompare: Wert mit dem die Eigenschaft verglichen werden soll

**IPropertyNotificator:** Der „IPropertyNotificator“ wird an das Person Model vererbt und dient dazu die UI über Änderungen am Model zu informieren, damit diese sich aktualisieren kann. Die Klasse beherbergt eine SetField Methode, die als Setter Funktion fungiert, jedoch ein Event auslöst, bei jedem Setzen des Events.

**Person:** „Person“ ist die Basisklasse für alle Models, die in der Anwendung vom Nutzer verwendbar sein sollen. Sie enthält die Grundlegendsten Eigenschaften einer Person. Alle Eigenschaften die in dieser Klasse definiert sind, sind auch filterbar.

**UniversityMember:** Eine Abstrakte Klasse, die als Basisklasse das Model Person verwendet und als Basisklasse für Lecturer und Student dient. Die Klasse soll nicht instanziiert werden können und ist dementsprechend auch nicht im PersonModelDictionary hinterlegt.

**Lecturer & Student:** Sind Klassen mit „UniversityMember“ als Basisklasse. Sie sind vom Nutzer Erstellbar, demensprechend im PersonModelDictionary hinterlegt und ebenfalls auf ihre eigenen Felder filterbar.

**ModelConverter:** Der „ModelConverter“ ist ein Service, der es ermöglicht eine Instanz in eine andere Instanz zu „konvertieren“ Dies geschieht durch Ermitteln der Eigenschaften der Source und Target Instanz. Überlappungen werden dann von der Source zum Target überschrieben. Anschließend wir die Target Instanz zurückgegeben.

## Referenzen:

Es wurden 2 NuGet Erweiterungen Genutzt.

* MoreLinq: für erweiterte Linqabfragen (FilterEntry Zeile 135: allProperties.**DistinctBy()**)
* Newtonsoft.Json zum serialisieren und deserialisieren der Daten

## Abgedeckte Themen:

* Inheritance and Polymorphism
* The ultimate base class object
* Abstract Classes
* Properties
* (Interfaces) => INotifyPropertyChanged verwendet
* Events
* Annonymous methods
* Lambda Functions (are anonymous methods)
* WPF-Application
* User-Controls
* Generic Types and Classes
* Collections
* List<T>
* Creating your own Collection with Collection<T>
* LINQ-Queries

## Eigene Themen:

* Databinding
* JSON serialisierung/deserialisierung
* Öffnen, speichern von Dateien
* Reflection
* WPF Code-Behind
* NuGet Extensions

## Mögliche zukünftige Erweiterungen:

Model erstellbar durch Nutzer mittels templates und automatisches Einbinden in das PersonModelDictionary durch scannen des ModelFolders.

## Eigeneinschätzung

