 **INSTYTUT PODSTAW INFORMATYKI POLSKIEJ AKADEMII NAUK**

***PROGRAMOWANIE NA PLATFORMIE .NET***

**Programowanie aplikacji samodzielnych (WPF) - Warsztat 2**

**WPF Tutorial**

<http://wpftutorial.net/>

**DataBinding in WPF**

<http://wpftutorial.net/DataBindingOverview.html>

**Free Online XPath Tester**

<http://www.freeformatter.com/xpath-tester.html>

**Przewodnik: moja pierwsza aplikacja klasyczna WPF - ExpenseIt**

<https://msdn.microsoft.com/library/ms752299(v=vs.110).aspx>

### Analiza działania ExpenseIt

<https://msdn.microsoft.com/library/ms752299(v=vs.110).aspx>

1. (WPF) Źródłem statycznych danych w formie XML jest XmlDataProvider - wspomniany w sekcji "Binding Data to a Control". Rozwiązanie to pozwala wczytać lokalnie zdefiniowaną część dokumentu XML, tzw. data island.

Korzeń wczytywanych danych określony jest atrybutem XPath, poniżej występuje znacznik <Expenses> oraz dalsze znaczniki i ich atrybuty:

<XmlDataProvider x:Key="**ExpenseDataSource**" XPath="**Expenses**">  
 <x:XData>  
 <**Expenses** xmlns="">  
 <**Person** Name="Mike" Department="Legal">  
 <Expense ExpenseType="Lunch" ExpenseAmount="50" />  
 <Expense ExpenseType="Transportation" ExpenseAmount="50" />  
 </Person>  
 ...  
 </Expenses>  
 </x:XData>  
</XmlDataProvider>

Znaczniki XML są dynamicznie zamieniane przez XmlDataProvider na ***obiekty***, na których można wykonywać dalsze "zapytania" wskazując kolejne wyrażenia XPath (obiekty takie mają właściwości odpowiadające strukturze XML, czyli atrybutom oraz zagnieżdżonym znacznikom - patrz 5. dalej)

Jest to wykorzystane w dalszej konfiguracji kontrolki listy:

<ListBox Name="peopleListBox" Grid.Column="1" Grid.Row="2"   
 ItemsSource="{Binding Source={StaticResource **ExpenseDataSource**},  
 XPath=**Person**}"

ItemTemplate="{StaticResource **nameItemTemplate**}">  
</ListBox>

Jak widać powyżej, <Expenses> zawiera znaczniki <Person>, czyli wszystko się dopełnia:

* strona ExpenseItHome.xaml zawiera definicję danych jako XML, oraz kontrolkę listy do ich wizualizacji,
* dla listy źródłem danych jest więc XmlDataProvider określony jako **ExpenseDataSource**, który ma wskazany jako korzeń swoich danych znacznik <Expenses> atrybutem XPath
* źródło danych dla listy wskazane jest w **ItemsSource** poprzez wiązanie danych do zasobu statycznego o nazwie ExpenseDataSource
* stamtąd elementy dla listy są wybierane przez wskazanie znaczników <Person> atrybutem XPath

Ostatnią rzeczą jest wskazanie jak lista ma interpretować elementy reprezentujące <Person>.

Zdefiniowany jest w tym celu szablon elementu danych **nameItemTemplate**:

<DataTemplate x:Key="**nameItemTemplate**">  
 <Label Content="{Binding XPath=@Name}"/>  
</DataTemplate>

który jest użyty w atrybucie ItemTemplate, w ostatniej linii definicji kontrolki ListBox powyżej.

2. (WPF) Strona ExpenseItHome.xaml zawiera statyczne dane XML, z czego wyrażenia XPath pozwalają wybrać pozdbiór tych danych. Używane w metodzie obsługi zdarzenia przycisku wyrażenie:  
“new ExpenseReportPage(this.peopleListBox.SelectedItem)” - tworzy nowy obiekt drugiej strony aplikacji, przekazując mu część tego zbioru danych:

private void Button\_Click(object sender, RoutedEventArgs e)

{

// View Expense Report

ExpenseReportPage expenseReportPage =

new ExpenseReportPage( **this.peopleListBox.SelectedItem** );

this.NavigationService.Navigate(expenseReportPage);

}

W tym przypadku jest to zaznaczony element danych listy, czyli obiekt równoważny znacznikowi <Person> wraz z zawartymi w nim znacznikami <Expense>:

<**Person** Name="Mike" Department="Legal">  
 <Expense ExpenseType="Lunch" ExpenseAmount="50" />  
 <Expense ExpenseType="Transportation" ExpenseAmount="50" />  
 </Person>

3. (WPF/C#) Z kolei konstruktor klasy ExpenseReportPage zapamiętuje przekazany obiekt ustawiając DataContext dla całej strony:

public ExpenseReportPage(object data) : this()

{

// Bind to expense report data.

this.DataContext = data;

}

Kontrolki w widoku mogą mieć określone źródło danych, domyślnie jest ono puste. **DataContext** jest właśnie sposobem na to, aby wskazać jakieś źródło danych - może to być inna kontrolka, zasób statyczny, kolekcja obiektów zdefiniowana w pliku C# Code-Behind dla strony/okna WPF - jest to bardzo elastyczne.

Można ustawić kontekst danych dla dowolnie zagnieżdżonej kontrolki, lub dla kontenera zawierającego inne kontrolki. Kontekst danych propaguje się w dół, do kontrolek potomnych, czyli wystarczy go ustawić dla strony (this.DataContext) aby móc używać wyrażeń wiązania danych w innych kontrolkach. Nie trzeba wtedy określać źródła danych dla każdej kontrolki osobno.

4. (C#) Wyrażenie " **: this()** " określa wywołanie konstruktora bezparametrowego "ExpenseReportPage()" w ramach tej samej klasy ExpenseReportPage. Można dzięki temu inicjować stan konstruowanego obiektu pisząc w konstruktorze bezparametrowym podstawowy sposób inicjalizacji, a w innych konstruktorach wywołując tą samą inicjalizację ORAZ wykonując później dodatkowe czynności. Unika się w ten sposób duplikacji kodu.

Poniższe części kodu są sobie równoważne:

public ExpenseReportPage()  
 {  
 **InitializeComponent();**  
 }

public ExpenseReportPage(object data)  
 {  
 **InitializeComponent();**  
 this.DataContext = data;  
 }

Przekazując sterowanie między wersjami konstruktorów unika się duplikacji wywołania InitializeComponent():

public ExpenseReportPage()  
 {  
 InitializeComponent();  
 }

public ExpenseReportPage(object data) **: this()**  
 {  
 this.DataContext = data;  
 }

Klasa może mieć oczywiście wiele konstruktorów, poniżej *hipotetyczny* przykład konstruktora przyjmującego

spersonalizowany tytuł strony:

public ExpenseReportPage()  
 {  
 InitializeComponent();  
 }

public ExpenseReportPage(object data) **: this()**  
 {  
 this.DataContext = data;  
 }

public ExpenseReportPage(string title) **: this(null)**  
 {  
 this.Title = title;  
 }

Trzeci konstruktor wywołuje drugi (z parametrem typu **object**), podając **null** w miejsce właśnie parametru **data**.

Drugi konstruktor wywołuje pierwszy (bezparametrowy), który wywołuje InistalizeComponent().

Tworząc więc obiekt przez new ExpenseReportPage("Raport") efektywna kolejność wykonania instrukcji to:

InitializeComponent();

this.DataContext = null;

this.Title = "Raport";

Zgodnie z regułami języka C# wywołanie **this(...ewentualne parametry...)** musi być pierwsze w liście inicjalizacyjnej konstruktora (po znaku ":").

5. (WPF) W obsłudze zdarzenia przycisku tworzona jest druga strona aplikacji:

private void Button\_Click(object sender, RoutedEventArgs e)  
{  
 // View Expense Report  
 ExpenseReportPage expenseReportPage =

new ExpenseReportPage( **this.peopleListBox.SelectedItem** );  
 this.NavigationService.Navigate(expenseReportPage);  
  
}

Wywołany jest przy tym konstruktor przyjmujący ***obiekt***, jako nietypowany "object data".

Obiekt ten wskazywany przez właściwość SelectedItem kontrolki listy odpowiada jednemu ze znaczników <Person> z data island na pierwszej stronie aplikacji.

Znacznik <Person> zawiera wydatki dla danej osoby - jako znaczniki <Expense> z atrybutami ExpenseType i ExpenseAmount.

Dalsze wiązania danych na drugiej stronie aplikacji robią całą resztę - wskazując przez **XPath** kolejne dopasowania do właściwości

przekazanego do tej strony obiektu, który zawiera część wczytanych z XML danych.

(a dokładnie do atrybutów znaczników XML - co widać po "@" + nazwa atrybutu)

Dla nagłówków - z <Person>:

Content="{Binding XPath=@Name}"

Content="{Binding XPath=@Department}"

Dla kolejnych elementów w siatce danych - z <Person> <Expense>:

Content="{Binding XPath=@ExpenseType}"

Content="{Binding XPath=@ExpenseAmount}"

#### 

#### 

#### ExpenseIt - Modyfikacja 1

Dane źródłowe XML dla listy w stronie ExpenseItHome.xaml wskazywane są przez nazwę ExpenseDataSource. Można przenieść definicję tych danych do miejsca ogólnie dostępnego, takiego jak App.xaml. W ten sposób danych tych można będzie używać wielokrotnie, na wielu stronach aplikacji.

1. Wyciąć z pliku ExpenseItHome.xaml linie:  
     
    <!-- Expense Report Data -->  
    <XmlDataProvider x:Key="ExpenseDataSource" XPath="Expenses">  
    ... ...  
    </XmlDataProvider>
2. Przenieść je do pliku App.xaml, do sekcji <Application.Resources>  
     
   **<Application.Resources>**  
    <!-- Expense Report Data -->  
    <XmlDataProvider x:Key="ExpenseDataSource" XPath="Expenses">  
    ... ...  
    </XmlDataProvider>  
     
   **</Application.Resources>**
3. Uruchomić ponownie aplikację.

#### 

#### 

#### ExpenseIt - Modyfikacja 2

Dane wskazywane przez wyświetlane są tylko w trakcie działania programu. W programie Blend oraz Visual Studio 2017 możliwe jest jednak wyświetlanie przykładowych danych w trybie edycji.

1. Dodać do projektu nowy plik XML o nazwie: **Expenses.xml**  
    Projekt > [menu kontekstowe] > Add > New Item > w oknie:  
    | Visual C# > XML File |
2. Wkleić do środka treść:  
     
   <?xml version="1.0" encoding="utf-8" ?>  
   <Expenses xmlns="">  
    <Person Name="Mike2" Department="Legal1">  
    <Expense ExpenseType="Lunch" ExpenseAmount="50" />  
    <Expense ExpenseType="Transportation" ExpenseAmount="50" />  
    </Person>  
    <Person Name="Lisa2" Department="Marketing1">  
    <Expense ExpenseType="Document printing" ExpenseAmount="50"/>  
    <Expense ExpenseType="Gift" ExpenseAmount="125" />  
    </Person>  
    <Person Name="John2" Department="Engineering1">  
    <Expense ExpenseType="Subscription" ExpenseAmount="50"/>  
    <Expense ExpenseType="New machine" ExpenseAmount="600" />  
    <Expense ExpenseType="Software" ExpenseAmount="500" />  
    </Person>  
    <Person Name="Mary2" Department="Finance1">  
    <Expense ExpenseType="Dinner" ExpenseAmount="100" />  
    </Person>  
   </Expenses>
3. We właściwościach pliku Expenses.xml ustawić:  
   Build Action: **Content**Copy To Output Directory: **Copy if newer**  
   Custom Tool: *{wyczyścić, jeśli nie jest puste}*
4. W pliku ExpenseItHome.xaml dodać tuż przed pierwszym znacznikiem <Grid> definicję:  
     
    <Page.Resources>  
    <XmlDataProvider x:Key="**xmlFileRes**"  
    Source="./Expenses.xml" XPath="Expenses" />  
    </Page.Resources>
5. Zamienić definicję kontrolki listy nazw pracowników na:  
     
   <ListBox Name="peopleListBox" Grid.Column="1" Grid.Row="2"  
    **ItemsSource="{Binding Source={StaticResource xmlFileRes},  
    XPath=Person}"** ItemTemplate="{StaticResource nameItemTemplate}" />
6. Otworzyć zmienioną stronę do edycji w środowisku Blend:  
   [Solution Explorer] > ExpenseItHome.xaml > [menu kontekstowe] > Design in Blend
7. Uruchomić aplikację z poziomu środowiska Blend
8. Zamknąć środowisko Blend

#### 

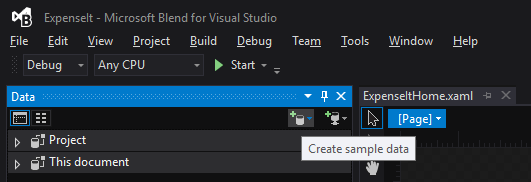
#### 

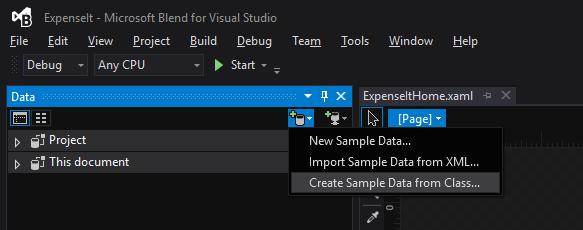
#### ExpenseIt - Modyfikacja 3

Wiązania danych można też określać, co zresztą często się robi, dla danych będących obiektami biznesowymi (obiektami CLR - środowiska wykonawczego). W tym celu trzeba utworzyć odpowiednie klasy, przygotować zbiory obiektów oraz zmienić wiązania danych.

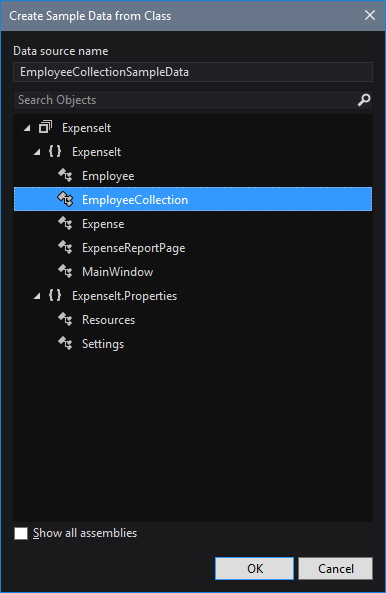
Ten ostatni krok jest potrzebny, ponieważ dla danych XMLDataProvider wyrażenie wybierające dane wskazywane określane jest przez **XPath**, zaś atrybuty ze znakiem @, np. **@Name**.  
Dla własnych obiektów zaś wyrażenie wybierające dane wskazywane jest po prostu przez **Path**, zaś atrybuty obiektu wprost przez nazwę właściwości (property), np. **Name**.

1. Dodać do projektu klasę C#: **Employee.cs**  
    Projekt > [menu kontekstowe] > Add > Class
2. Wypełnić plik poniższą treścią:  
      
   **using** System.Collections.ObjectModel; //dla ObservableCollection  
     
   namespace ExpenseIt  
   {  
    // The Expense class defines a simple Expense business object.  
    **class Expense** {  
    public Expense() { }  
     
    public string ExpenseType { get; set; }  
    public double ExpenseAmount { get; set; }  
    }  
     
    // The Employee class defines Employee business object.  
    **class Employee**  
    {  
    public Employee()  
    {  
    this.Expense = new List<Expense>();  
    }  
     
    public string Name { get; set; }  
    public string Department { get; set; }  
     
    public List<Expense> Expense { get; }  
    }  
     
    // The EmployeeCollection class defines a simple collection  
    // for Employee business objects.  
    **class EmployeeCollection** : ObservableCollection<Employee>  
    {  
    public EmployeeCollection() { }  
    }  
   }
3. W pliku Code-Behind strony ExpenseHome.xaml.cs dodać:  
     
    public partial class ExpenseItHome : Page  
    {  
    **private static EmployeeCollection collection;**  
    public ExpenseItHome()  
    {  
    InitializeComponent();  
     
    **if (collection == null)**  **PrepareCollection();  
    DataContext = collection;** }  
     
    **private void PrepareCollection()**  
    {  
    **collection = new EmployeeCollection();  
    Employee employee = new Employee {  
    Name = "Wall-E", Department = "BnL Earth" };  
    employee.Expense.Add(  
    new Expense { ExpenseType = "Driving belt",  
    ExpenseAmount = 2.0 });  
    employee.Expense.Add(  
    new Expense { ExpenseType = "Light bulb",  
    ExpenseAmount = 1.0 });  
    collection.Add(employee);  
    collection.Add(new Employee  
    {  
    Name = "Eve",  
    Department = "BnL Axiom",  
    Expense = {  
    new Expense {  
    ExpenseType = "Green plant",  
    ExpenseAmount = 1.0 }  
    }  
    });** }
4. W pliku strony ExpenseHome.xaml dodać definicję szablonu elementu listy **nameItemTemplateEmployee**:  
     
    <!-- Name item template -->  
    <DataTemplate x:Key="nameItemTemplate">  
    <Label Content="{Binding **XPath=@Name**}"/>  
    </DataTemplate>  
    **<!-- Name item template for Employee -->**  
    **<DataTemplate x:Key="nameItemTemplateEmployee">  
    <Label Content="{Binding Path=Name}"/>  
    </DataTemplate>**
5. Zmodyfikować definicję listy osób:  
     
   <ListBox Name="peopleListBox" Grid.Column="1" Grid.Row="2"  
    **ItemsSource="{Binding}"  
    ItemTemplate="{StaticResource nameItemTemplateEmployee}">**  
   </ListBox>
6. Uruchomić aplikację - powinny wyświetlić się dane z obiektów klasy Employee,  
   tworzone w konstruktorze strony w pliku ExpenseHome.xaml.cs
7. Zamknąć aplikację
8. Otworzyć stronę ExpenseItHome.xaml w środowisku Blend:  
   [Solution Explorer] > ExpenseItHome.xaml > [menu kontekstowe] > Design in Blend
9. Otworzyć widok źródeł danych projektu:  
   Blend [Menu główne] > View > Data Window
10. Wygenerować przykładowe dane dla własnych obiektów:  
    Przycisk “Create sample data” > polecenie “Create Sample Data from Class...”

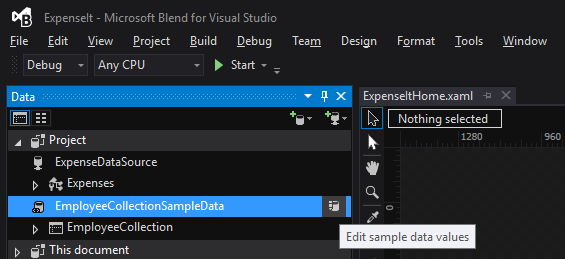




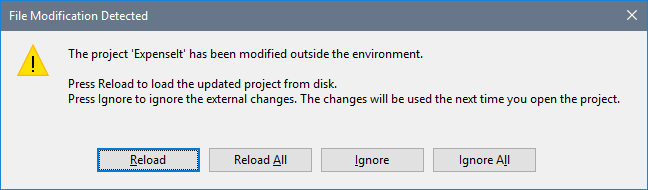
1. Jako typ danych przykładowych obiektów wskazać **EmployeeCollection**:



1. W projekcie pojawi się nowe źródło danych: **EmployeeCollectionSampleData**
2. Można sprawdzić wygenerowaną zawartość wciskając przycisk “Edit sample data values”



1. Zamknąć środowisko Blend, lub wybrać polecenie:  
   Blend [Menu główne] > File > Save All (Ctrl+Shift+S)
2. Powrócić do środowiska Visual Studio.
3. Potwierdzić przeładowanie zmienionego projektu - przyciskiem “Reload”:



1. W drzewie projektu będzie widoczny dodatkowy katalog **SampleData**, zawierający plik XAML z przykładowymi danymi: **EmployeeCollectionSampleData.xaml**
2. Otworzyć stronę ExpenseItHome.xaml
3. Wskazać na stronie kontekst danych dla trybu projektowania, korzystając z rozszerzenia XAML d:DataContext - jak poniżej:

<Page x:Class="ExpenseIt.ExpenseItHome"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:local="clr-namespace:ExpenseIt"

mc:Ignorable="d"

d:DesignHeight="500" d:DesignWidth="640"

**d:DataContext="{d:DesignData /SampleData/EmployeeCollectionSampleData.xaml}"**

Title="ExpenseIt - Home">

1. W trybie edycji powinny pojawić się dane przykładowe.
2. Analogicznie strona otwarta w środowisku Blend powinna zawierać dane przykładowe.
3. Zaś po uruchomieniu aplikacji - powinny pojawić się dane z obiektów klasy Employee.
4. Na koniec można jeszcze zmienić obrazki tła dla stron aplikacji. W tym celu trzeba dodać do projektu pliki graficzne i zmienić wartości ImageSource, jak poniżej:

- w pliku ExpenseItHome.xaml:

<Grid.Background>

<ImageBrush **ImageSource**=**"wall-e.png"**/>

</Grid.Background>

- w pliku ExpenseReportPage.xaml:

<Grid.Background>

<ImageBrush **ImageSource**=**"wall-e-eve.png"**/>

</Grid.Background>

### Pokazywanie przykładowych danych w trybie projektowania - Blend

**Walkthrough: Using Sample Data in the WPF Designer**

<https://msdn.microsoft.com/library/ee823176(v=vs.100).aspx>

**WPF: Simulating data in design mode in Microsoft Expression Blend**

<http://www.galasoft.ch/mydotnet/articles/article-2007091401.html>

**CollectionView - How to Navigate, Group, Sort and Filter Data in WPF**

<http://wpftutorial.net/DataViews.html>

#### 

#### 

#### ExpenseIt - Modyfikacja 4

Pozostało jeszcze zmodyfikować stronę **ExpenseReportPage.xaml**, aby także używała nowych wiązań danych do obiektów CLR (zamiast do danych XML).

1. Otworzyć stronę ExpenseReportPage.xaml
2. Zamienić istniejące wiązania danych do atrybutów i elementów XML, wskazując wiązania do obiektów CLR i ich właściwości.  
   W tym celu wystarczy zamienić **XPath** na **Path**, oraz usunąć określenia atrybutów **@**:

<Grid.Resources>

<!-- Reason item template -->

<DataTemplate x:Key="typeItemTemplate">

<Label **~~Content="{Binding XPath=@ExpenseType}"~~**

**Content="{Binding Path=ExpenseType}"**/>

</DataTemplate>

<!-- Amount item template -->

<DataTemplate x:Key="amountItemTemplate">

<Label **~~Content="{Binding XPath=@ExpenseAmount}"~~**

**Content="{Binding Path=ExpenseAmount}"**/>

...

<!-- Name -->  
 <Label Style="{StaticResource labelStyle}"  
 **~~Content="{Binding XPath=@Name}"~~**  
 **Content="{Binding Path=Name}"**></Label>  
 <!-- Department -->  
 <Label Style="{StaticResource labelStyle}"  
 **~~Content="{Binding XPath=@Department}"~~**  **Content="{Binding Path=Department}"**></Label>  
  
 <!-- Expense type and Amount table -->  
 <DataGrid  
 **~~ItemsSource="{Binding XPath=Expense}"~~**  **ItemsSource="{Binding Path=Expense}"**  
  
 <DataGrid.Columns>  
 <DataGridTextColumn Header="ExpenseType"  
 **~~Binding="{Binding XPath=@ExpenseType}"~~  
 Binding="{Binding Path=ExpenseType}"** />  
 <DataGridTextColumn Header="Amount"  
 **~~Binding="{Binding XPath=@ExpenseAmount}"~~  
 Binding="{Binding Path=ExpenseAmount}"** />  
 </DataGrid.Columns>

1. Uruchomić aplikację, wybrać pracownika i wcisnąć przycisk wyświetlający widok raportu.
2. Powinny zostać wyświetlone szczegóły z obiektu klasy Employee.

#### ExpenseIt - Modyfikacja 5

Na koniec można jeszcze dodać przycisk tworzący nowe obiekty danych w kolekcji.

1. Otworzyć stronę ExpenseItHome.xaml
2. Wstawić pod istniejącym, nienazwanym przyciskiem drugi przycisk:

<!-- View report button -->

<Button Grid.Column="1" Grid.Row="3" Click="Button\_Click"

Style="{StaticResource buttonStyle}">View</Button>

**<!-- Add employee button -->**

**<Button x:Name="buttonAdd" Content="Add Employee"**

**Grid.Column="1" Grid.Row="3"**

**HorizontalAlignment="Left"**

**VerticalAlignment="Top" Width="120" Height="25"**

**Margin="0,10,0,0" Click="buttonAdd\_Click"/>**

1. Wygenerować metodę obsługi zdarzenia **Click** klikając na przycisku dwukrotnie.  
   Powinien pojawić się edytor pliku ExpenseItHome.xaml.cs:

private void buttonAdd\_Click(object sender, RoutedEventArgs e)

{

}

1. Uzupełnić kod metody następująco:

private void buttonAdd\_Click(object sender, RoutedEventArgs e)

{

**Employee employee = new Employee**

**{**

**Name = "Employee " + (collection.Count + 1),**

**Department = "Random"**

**};**

**collection.Add(employee);**

**DataContext = null;**

**DataContext = collection;**

}

1. Uruchomić aplikację.
2. Wciśnięcie przycisku “Add Employee” powinno dodawać nowy element i odświeżać zawartość listy.
3. Tak dodany obiekt nie będzie jednak mieć żadnych wydatków (Expense).
4. Uzupełnij kod tworzący wydatki na jeden z dwóch sposobów, analogicznie jak w konstruktorze strony: public ExpenseItHome() - patrz: Modyfikacja 3, punkt 3.

### 

### Do czytania

**Introduction to WPF**

<https://msdn.microsoft.com/library/mt149842(v=vs.110).aspx>

**Creating a UI by using Blend for Visual Studio**

<https://msdn.microsoft.com/library/jj171012.aspx>

**Styling and Templating**

<https://msdn.microsoft.com/library/ms745683(v=vs.110).aspx>

**Data Templating Overview**

<https://msdn.microsoft.com/library/ms742521(v=vs.110).aspx>

**Commanding Overview**

<https://msdn.microsoft.com/library/ms752308(v=vs.100).aspx>