Упражнение №4 по ПС - Част 1

Data Binding

Binding
DataContex
Data Template
INotifyPropertyChanged
ObservableCollection

Целта на това упражнение:
Запознаване с концепцията за Data Binding и неговата реализация в WPF.

Необходими познания:
От предметите дотук и предното упражнение трябва да можете:

- Да може да сглобите основно визулано приложение
- Да може да опишете основен интерфейс със XAML
- Да познавате основните контроли в WPF

Легенда:

Най-важното

- Схематично и накратко

Обяснение. Уточнение.

Пример: или **Задача:** ограденото трябва да изпълните за да е синхронизиран проекта ви

- ➢ Подходящо място да стартирате изпълнение за да проверите дали работи вярно.

Задачите в упражнението изграждат:
Приложение за следене на разходи

В това упражнение: Разширяваме упражнително приложение за раходи
- прозорецът със списъка на всички хора се променя, така че списъка да е свързан към списък с обекти хора, като всеки обект-човек съдържа в себе си и раходите на човека
- втора страница поличава за контекст избрания обект-човек, а контролите на нея са свързани към описаните разходи в обекта-човек-контекст.
- добавяме свойство и контрола свързана към него, което описва последна интеракци с приложението. Промяна на свойството опреснява и контролата.

В края на упражнението:
Потребителят вижда прозорец със списък на всички хора с разходи и бутон за преминаване към друга страница, в която се извеждат на разходите на избрания човек.

За синхронизация на упражненията:

Добавете в проекта **StudentRepository**.

- Потребителя да разполага с две полета, в които да въведе потребителско име и парола
- Потребителя да разполага с бутон, с който да стартира проверката на въведените потребителско име и парола за вписване (LogIn)
- При успешно вписване на потребител от тип СТУДЕНТ да се активират полетата за данни на студент и да се попълнят с неговите данни.

- Потребителят да разполага с бутон, с който да се отпише от системата (Logout)

1. Binding (Свързване)

Data binding = general technique that binds two data/information sources together and maintains synchronization of data.

Свързване (Data binding) най-често се използва за връзка на интерфейса с данни от кода.
Така са спестява код за синхронизация и циклене по елементи на списъци.

1.1. Синтаксис

Свързването работи, като вместо стойност (на свойство) посочим връзката в къдрави скоби с ключовата дума Binding:

{Binding}

Обикновено уточняваме връзката към конкретно свойство на обект:

{Binding Path=NameOfProperty}

Връзката Binding има и други полета за настройка, освен Path, например:

{Binding Path=Text, ElementName=txtValue}

Свързването работи единствено към публични свойства.

```
Content="{Binding Path=SelectedItem.Content, ElementName=peopleListBox}
```

1.2. Свързване към поле

За да можем да свършем **свойство** на контрола **към друг обект**, трябва да можем да укажем кой е този обект.
Най-лесно може да определм обект, намиращ се в същия XAML маркъп. За това служи `ElementName`:

В проекта **ExpenseIt:**

На направим бутона „View” ни подсеща кой ред ще прегледаме, като подменим свойството му Content, така че да е свързано към избраното име от списъка в Lisbox
:

1. Преместете съдържанието на бутона, ако е зададено между таговете, като свойство Content в отварящия таг:

```
<Button ... >View</Button>      →→→      <Button ... Content="View"></Button>
```

2. Променете свойството Content да е свързано към друга контрола

```
Content="{Binding ElementName }
```

3. Задайте контролата да е `peopleListBox`:

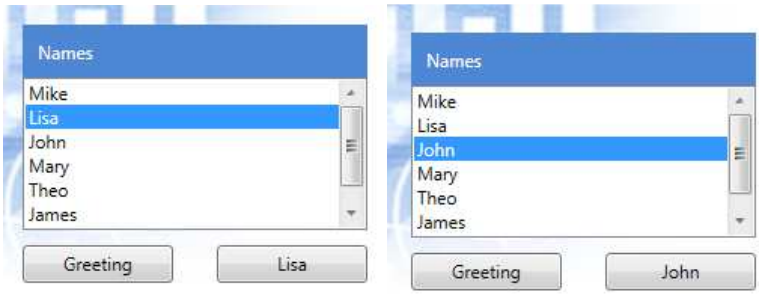
```
ElementName=peopleListBox
```

4. Дотук компилаторът ще посочи, че не може да се свържем към цялата контрола (The property 'Content' cannot be data bound to a visual element.), за това трябва да посочим от кое точно свойство ще черпим информация чрез `Path`:

   4.1. Добавете `, Path=` в скобите за свързване.
   4.2. Искаме да се свържем с текущия избран обект: `SelectedItem`
   4.3. Обектите ни в ListBox са ListBoxItem, т.е. видимата им част, към която трябва да се свържем е Content: `SelectedItem.Content`

```
{Binding ElementName=peopleListBox, Path=SelectedItem.Content}
```

Резултатът: Надписът на бутона ще е еднакъв с написът на избрания елемент от списъка. При зареждане `SelectedItem` няма, за това и бутонът ще е без надпис.



1.3. Свързване към списък

За да можем да свършем **списъчно свойство** на контрола **към друг списък**, трябва да можем да укажем кой е този списък.
Най-лесно може да определм списък, видим в същия XAML маркъп по име, като ресурс `StaticResource`:

Достъпа до възлите в ресурсите е с малко по-различен синтаксис:
- До конкретен възел с `XPath` (вместо `Path`)
- До атрибут на възел с @ (вместо точка-свойство)

1. Отворете **ExpenseItHome.xaml**, и след отварящия **Grid** елемент, добавете следния XAML код за създаването на **XmlDataProvider**, който ще съдържа данните за всеки един човек.

В случая ще зададем данните като ресурс на Grid-а, но обикновено те ще трябва да се четат от файл или база данни.

```
<Grid.Resources>
    <!-- Expense Report Data -->
    <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>
                <Person Name="Lisa" Department="Marketing">
                    <Expense ExpenseType="Document printing" ExpenseAmount="50"/>
                    <Expense ExpenseType="Gift" ExpenseAmount="125" />
                </Person>
                <Person Name="John" Department="Engineering">
                    <Expense ExpenseType="Magazine subscription" ExpenseAmount="50"/>
                    <Expense ExpenseType="New machine" ExpenseAmount="600" />
                    <Expense ExpenseType="Software" ExpenseAmount="500" />
                </Person>
                <Person Name="Mary" Department="Finance">
                    <Expense ExpenseType="Dinner" ExpenseAmount="100" />
                </Person>
                <Person Name="Theo" Department="Marketing">
                    <Expense ExpenseType="Dinner" ExpenseAmount="100" />
                </Person>
            </Expenses>
        </x:XData>
    </XmlDataProvider>
</Grid.Resources>
```

2. Добавете данни за `James` и `David`.

3. Изтрийте `<ListBoxItem>` таговете от ListBox-а, защото вече ще ги черпим от списъчния ресурс на грида.
   Изтрийте и кода от конструктора, който добавя допълнителни редове.

4. Добавете следният атрибут към ListBox-а.

```
ItemsSource="{Binding Source={StaticResource ExpenseDataSource}}"
```

   Стартирайте програмата с **F5** или зеления триъгълник.
   Можете да видите, че в списъка има само 1 елемент без текст в него. Това е така, защото е приело целия <x:XData> таг като един голям обект и е изброило само него.

5. Трябва да укажем, че искаме хората от ресурса. Добавете към `Binding`-а (между скобите { }), разделено със запетайка от `Source` :

```
, XPath=Person
```

   Стартирайте програмата с **F5** или зеления триъгълник.
   В списъка вече има 7 елемента без текст в тях. Това са 7-те човека, но не е ясно кое от техните свойства трябва да се визуализира .

Бутонът „View” вече не извежда името на избрания елемент от ListBox.
Това е така, защото `SelectedItem` вече няма свойство `Content`, защото вече не е от тип `ListBoxItem`, а от тип `XmlElement`.

5. Променете пътя (`Path`) на свързването (`Binding`) на /бившия/ бутон „View” да се обръща към атрибута `Name`:

```
SelectedItem.Attributes[Name]
```

Стартирайте програмата с **F5** или зеления триъгълник.

В списъка все още има 7 елемента без текст в тях, но името на селектирания се появява в бутона.

2. DataContex

За да можем да свършем **свойство** на контрола **към свойство на произволен друг обект**, трябва да можем да укажем кой е този обект.
Ако не използваме изрично друга контрола или ресурс, трябва да посочим контекст, в рамките на който да се търси посоченото за свързване свойство или обект. За това служи `DataContex`:

1. Отворете **ExpenseReportPage.xaml.cs**. и добавете конструктор, който приема параметър обект, с който да подадете информация за разходните данни на избрания човек.

```
// Custom constructor to pass expense report data
public ExpenseReportPage(object data)
    : this()
{
    // Bind to expense report data.
    this.DataContext = data;
}
```

2. В ExpenseItHome.xaml.cs променете метода на клик събитието, така че да извиква новия конструктор, предавайки данните за избрания човек.

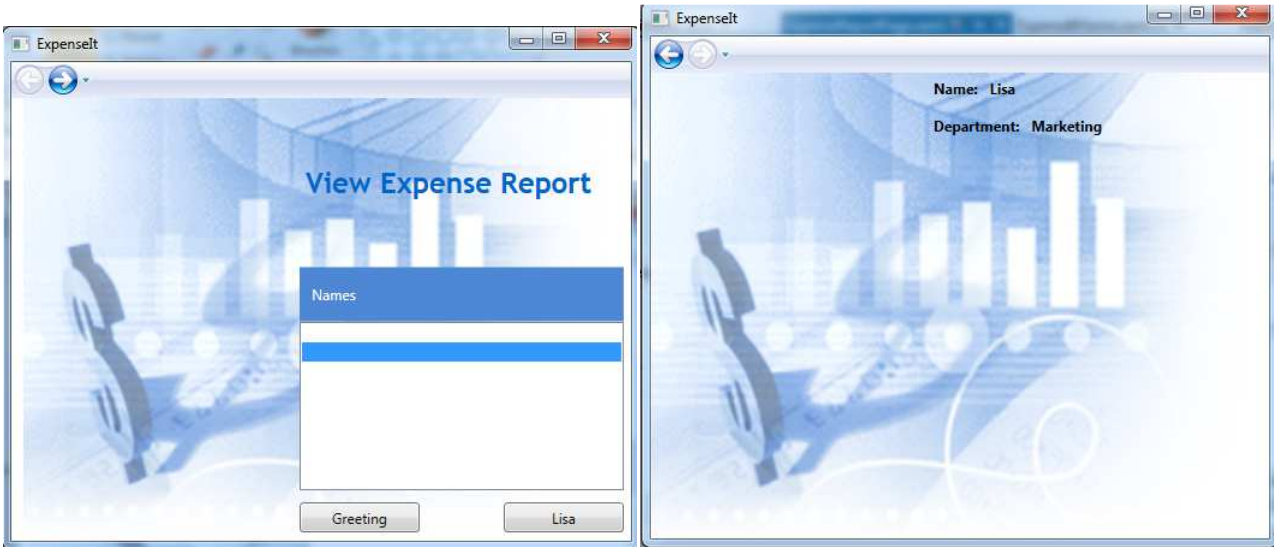
```
private void Button_Click(object sender, RoutedEventArgs e)
{
    // View Expense Report
    //ExpenseReportPage expenseReportPage = new ExpenseReportPage();
    //this.NavigationService.Navigate(expenseReportPage);
ExpenseReportPage expenseReportPage =
        new ExpenseReportPage(this.peopleListBox.SelectedItem);
    this.NavigationService.Navigate(expenseReportPage);
}
```

3. Отворете ExpenseReportPage.xaml, свържете (bind) съдържанието (content) на нов Label за елемента "Name":

```
<!-- Name -->
            <Label>Name:</Label>
            <Label Content="{Binding XPath=@Name}"></Label>
```

Свързваме директно към @Name атрибута, защото за DataContext на цялата страница сме подали обект Person.

4. Добавете още един нов свързан Label за елемента "Department"



2.1. Свързване към свойства на класа на формата

За да можем да виждаме свойства на прозорец /страница/, които не са контроли, трябва да посочим самия обект на прозореца /страницата/ като DataContext:

5. Добавете публично свойство в CodeBehind на класа `ExpenseItHome`

```
public DateTime LastChecked { get; set; }
```

6. Задайте стойност на свойството в конструктора

```
LastChecked = DateTime.Now;
```

7. Задайте в конструктора DataContext на обекта да е самия обект

```
this.DataContext = this;
```

8. Добавете Label свързан към свойството

```
<Label Grid.Column="0" Grid.Row="3" Content="{Binding Path=LastChecked}"/>
```

3. Data Templates

Свойството `ItemsSource`, което свързахме към списъка с хора `Expenses`, представлява списък, но за всеки елемент от него трябва да посочим как да се визуализира.

Т.е. трябва да посочим шаблон за извеждане чрез `DataTemplate`

Отворете **ExpenseItHome.xaml**

1. Да посочим какво да извежда за всеки ListBoxItem:
   1.1. Добавете в `</ListBox>` тага следни вложен таг:

```
<ListBox.ItemTemplate> </ListBox.ItemTemplate>
```

   1.2. Добавете в `<ListBox.ItemTemplate>` тага следния (двойно) вложен таг:

```
<DataTemplate> </DataTemplate>
```

   1.3. Добавете в `<DataTemplate>` тага един Label:

```
<Label />
```

   1.4. Добавете атрибут `Content=""` на Label-а
   1.5. Задайте му да е свързан към името

```
{Binding XPath=@Name}
```

Стартирайте програмата с **F5** или зеления триъгълник.
Този XAML свързва (binds) ItemsSource свойството на ListBox с източника на данни (data source) и прилага шаблона на данните (data template) като ItemTemplate.

3.1. DataGrid

На първата страница използвахне ListBox, защото е по-подходяща контрола за избор от потребителя.

Разходите на втората страница, можем също да покажем с Listbox и ItemTemplate с множество Label контроли.
Но може и да използваме контрола пригодена за извеждане на множество стойности от един обект, като DataGrid.

Отворете **ExpenseReportPage.xaml**

1. Добавете DataGrid за извеждане на разходите на всеки човек.

```
<DataGrid AutoGenerateColumns="False" RowHeaderWidth="0" >
    <DataGrid.Columns>
        <DataGridTextColumn Header="ExpenseType" />
        <DataGridTextColumn Header="Amount" />
    </DataGrid.Columns>
</DataGrid>
```

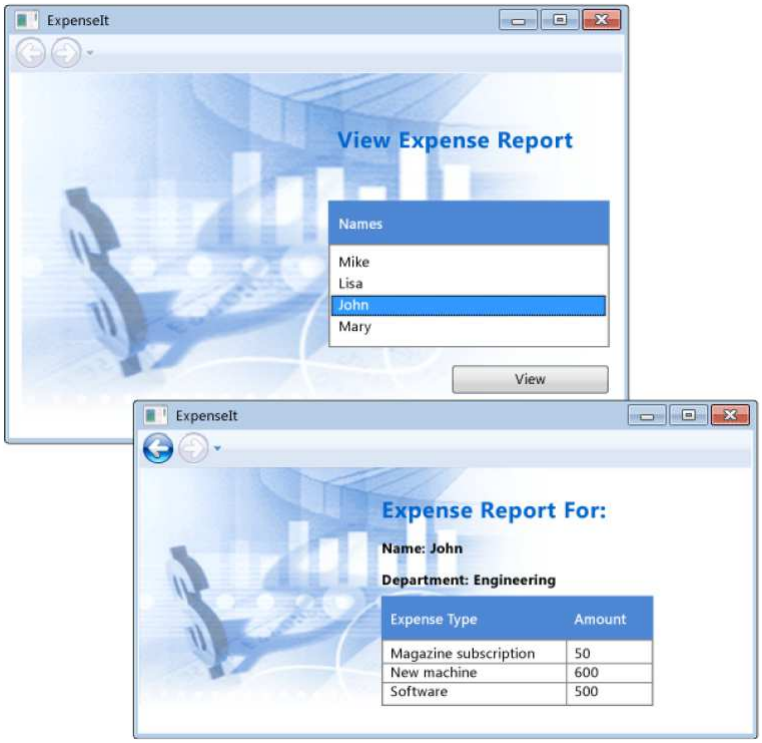
2. За да се приложат горните темплейти към колоните на DataGrid, трябва да се добавят следните връзки, маркирани в жълто.

```
<DataGrid Grid.Column="1"  Grid.Row="0" ItemsSource="{Binding XPath=Expense}"
```

```
            AutoGenerateColumns="False" RowHeaderWidth="0" >
    <DataGrid.Columns>
        <DataGridTextColumn Header="ExpenseType"
                            Binding="{Binding XPath=@ExpenseType}" />
        <DataGridTextColumn Header="Amount" Binding="{Binding XPath=@ExpenseAmount}"/>
    </DataGrid.Columns>
</DataGrid>
```

3. Стартирайте приложението, изберете човек и натиснете **View** бутона.

Програмата заедно с двата ѝ прозореца трябва да изглежда по подобен начин:



4. Промени

Промените в свързаните свойства не се отразяват автоматично на визуалните контроли след като веднъж са заредени при отварянето на прозореца.
Ако се отразяваха, това би означавало, че контролите проверяват постоянно дали е настъпила промяна в съдържанието на свойството, което би изисквало огромен изчислителен ресурс.

Вместо това трябва да се уведоми за настъпилата промяна чрез създаване на събитие.

Някои от готовите класове вече имат реализирана тази функционалност, за това връзката между бутона и списъка работи автоматично.
Когато свързваме към наши свойства, обаче, трябва се добави функционалността, ако важно да се отразяват промените.

4.1. Промени в свойство

Отворете **ExpenseItHome.xaml**

1. Добавете функция, която да се извиква при промяна на селектирания елемент в списъка `peopleListBox` :
   1.1. В Markup на `peopleListBox` добавете:

```
SelectionChanged="peopleListBox_SelectionChanged_1"
```

   1.2. В CodeBehind добавете:

```
private void peopleListBox_SelectionChanged_1(object sender,
          SelectionChangedEventArgs e)
{
    LastChecked = DateTime.Now;
}
```

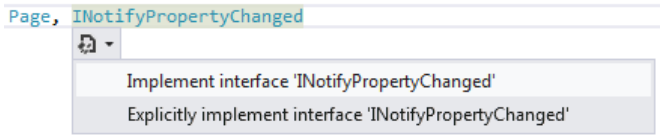
   Стартирайте програмата с **F5** или зеления триъгълник.
   Селектиране на друг ред променя стойността в свойството `LastChecked`*, но промяната не се отразява на интерфейса. Трябва да имплементираме INotifyPropertyChanged.*
Имайте предвид да изчакате, ако при вас изписва само час и минута.

2. В CodeBehind накарайте класа `ExpenseItHome` да наследява и от интерфейсът `INotifyPropertyChanged`:

```
using System.Collections.Generic;

public partial class ExpenseItHome : Page, INotifyPropertyChanged
```

3. Имплементирайте полета, които интерфейсът изисква



   3.1. Това ще добави поле от тип `PropertyChangedEventHandler`, който е делегат:

```
public event PropertyChangedEventHandler PropertyChanged;
```

4. Ще извикаме функцията-делегат `PropertyChanged` в set-ъра на свойството `LastChecked`, за да уведомим, че то има нова стойност. За целта:
   4.1. Трбва да променим set-ъра да има тяло, за което:
   4.2. Трябва да имаме частно поле от същия тип

```
private DateTime lastChecked;
public DateTime LastChecked
{
    get { return lastChecked; }
    set
    {
        lastChecked = value;
        // Извикване на PropertyChanged
    }
    }
```

5. Извикването на изисква два параметъра:

5.1. Обектът предизвикващ събитието `object sender`, за което можем да подадем `this`
5.2. И параметър `PropertyChangedEventArgs` , с който да укажем името на променящото се свойство, т.е. `"LastChecked"`:
5.3. Трябва и да има проверка за null, защото преди да се осъществи свързването (Binding) няма записан метод, който да слуша за събитието

```
if (PropertyChanged != null)
   PropertyChanged(this, new PropertyChangedEventArgs("LastChecked"));
```

4.2. Промени в списък

Уведомяване за промени в списък може да се изплементира както и за свойство (свойство от тип списък).

Има и готов списъчен тип, който сам реализира тази функционалност. Това е

```
using System.Collections.ObjectModel;
```

`ObservableCollection<T>`

(използвате вместо `List<T>`)

Отворете **ExpenseItHome.xaml**

Добавете списък на последно разгледаните хора:

1. В класа на страницата добавете свойство от тип `ObservableCollection` съхраняващо списък от низове `string`. Кръстете го `PersonsChecked`.
2. Свойството няма нужда от тяло, може да използвате късия синтаксис:

```
public ObservableCollection<string> PersonsChecked
{ get; set;}
```

3. Добавете и инстанциране в конструктора на калса:

```
PersonsChecked = new ObservableCollection<string>();
```

4. Да добавим в метода обработващ промяна на избрания човек `peopleListBox_SelectionChanged_1` да добавя името на избрания човек и в списъка от тип `ObservableCollection`:

```
PersonsChecked.Add(peopleListBox.SelectedItem.ToString());
```

5. Добавете Listbox вляво на страницата, в който да извеждаме селектираните хора:

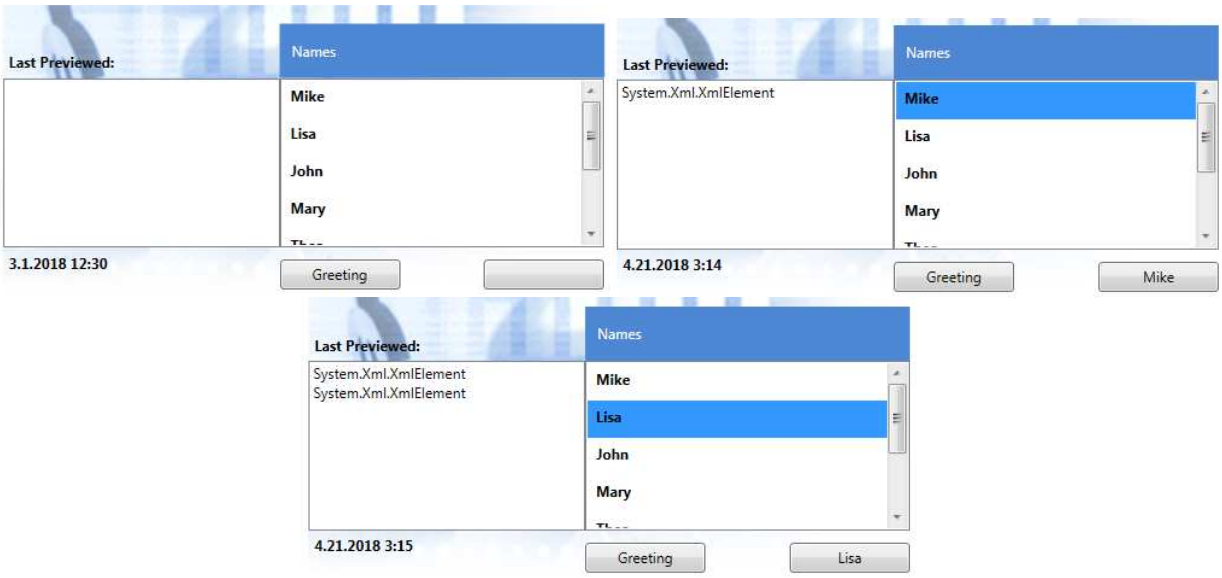
```
<ListBox Grid.Column="0" Grid.Row="2"></ListBox>
```

6. Задайте съдържанието на новия ListBox да е свързано със свойството `PersonsChecked` :

```
ItemsSource="{Binding Path=PersonsChecked}"
```

Стартирайте програмата с **F5** или зеления триъгълник.

При избиране на хора от списъка се попълва и другия списък на интерфейса, БЕЗ да сме използвали ръчно интерфейса `INotifyPropertyChanged`.
Само че към момента с неправилни стойности. Следва да го поправим.



7. Както вече уточнихме `SelectedItem` вече е от тип `XmlElement`.:
   7.1. Реално ни трябва същата стойност, която се подава и на бутона:
   ```
   PersonsChecked.Add(peopleListBox.SelectedItem.Attributes[Name]);
   ```
   7.2. Само че за разлика от свързването на интерфейса, което е предвидено да е гъвкаво, C# е строго типизиран език:
   ```
   'object' does not contain a definition for 'Attributes' and no extension method 'Attributes'
   ```
   7.3. Трябва да укажем от кой тип е `SelectedItem` :
   ```
   PersonsChecked.Add((peopleListBox.SelectedItem as
                                   System.Xml.XmlElement).Attributes["Name"])
   ```
   7.4. Атрибутът `Name` също не се интерпретира директно, трябва да се обърнем към свойството му `Value`:

```
PersonsChecked.Add((peopleListBox.SelectedItem as
                                   System.Xml.XmlElement).Attributes["Name"].Value);
```

Стартирайте програмата с **F5** или зеления триъгълник.

Вече попълва списъка с правилни стойности.

Общи задачи (пример за работа в час):

В проекта **StudentInfoSystem:**

Задача: Създайте следната функционалност:

- На страницата/прозореца с контроли за извеждане на данни за студент:
  - Контролите за данни на стдудент да са свързани към свойство от тип Student.
  - При вписване (Login) да се променя стойността на свойството.
  - Би следвало да може да изтриете метода за извеждане на данните на студент в контролите на формата без промяна във функционалността

Запишете си проекта! USB, DropBox, GoogleDrive, e-mail, SmartPhone, където и да е.
До края на упражненията ще работите върху този проект.
Вкъщи също ще работите върху този проект.
Накрая трябва да го представите завършен.