Agent Assignment Lab

Kommentarer til løsningsforslag

Lab1

Der sværeste i denne opgave er nok at vælge hvordan man vil angive databinding-source. Jeg har valgt at sætte vinduets Datacontext i koden for konstruktoren til en instans af agentklassen som oprettes i koden.

For alle relevante kontroller skal jeg så binde deres Text property til relevant property på Agentklassen. F.eks.:

```
Text="{Binding Path=ID}"
```

Lab2

I denne opgave har jeg valgt at sætte Datacontext på Grid'et i stedet for på vinduet (det er mest for at vise at det kan man også, det kræver blot at man har givet grid'et et Name i XAML-koden):

```
agentGrid.DataContext = agents;
```

Det store valg i denne opgave er hvordan man vil binde de enkelte kontroller til listen af agenter, og hvordan man vil synkronisere selectedItem i listboksen med det der vises i de individuelle kontroller. Da jeg har sat DataContext til listen af agenter, så kan det gøres meget smart ved at binde listboksens ItemSource (uden at angive Path, da den skal bindes til selve objektet i listen) samt sættes dens property IsSynchronizedWithCurrentItem til True.

```
<ListBox ItemsSource="{Binding}"
         IsSynchronizedWithCurrentItem="True"
         DisplayMemberPath="CodeName"
```

Alternativ kunne jeg have gjort som MacDonald og sat listboksens ItemSource i koden, og så i Xaml-koden bundet Grid'et til ListBoxens SelectedItem via elementBinding.

Lab3

I denne opgave er der 3 udfordringer:

1. Man skal sætte DataContext i XAML-koden,
2. Man skal kunne navigerer frem og tilbage i listen ved tryk på en knap og

Man skal kunne oprette nye agenter.

1: Agentklassen er forberedt for den første udfordring. Problemet er at man ikke kan instantiere parameteriserede klasser (bruge generics) i Xaml. Der er derfor lavet en wrapper klasse Agents som bruges i Xaml-koden. Det kan dog være en fordel at oprette nogle default data i klassens default konstruktør. Bemærk at designeren kalder denne, så man også i design-mode kan se data i vinduet.

```
<Window.Resources>
    <local:Agents x:Key="agents" /> ←laver en instans af Agent-klassen
</Window.Resources>
<Grid Name="agentGrid" DataContext="{StaticResource agents}">
```

2: I knappernes eventhandlere implementerer jeg navigeringen ved at ændre SelectedIndex for listboksen.

```
void btnForward_Click(object sender, RoutedEventArgs e)
    {
        if (lbxAgents.SelectedIndex < lbxAgents.Items.Count - 1)
            lbxAgents .SelectedIndex = ++lbxAgents.SelectedIndex;
    }
```

3: Det store problem med at indsætte en ny agent er, at listen er oprettet i Xaml-koden, og at man ikke kan give den et navn, således at den kan tilgås som et almindeligt datamedlem i koden. Jeg må derfor bruge FindResource og søge på nøglen. Det fundne objekt kan så typecastes til Agents og vi kan tilføje vores nye agent. Af hensyn til brugervenlighed flytter jeg selected index til den nye agent og flytter focus til det første indtastningsfelt.

```
Agents agents = (Agents)this.FindResource("agents");
agents.Add(new Agent());
lbxAgents.SelectedIndex = lbxAgents.Items.Count - 1;
tbxId.Focus();
```

Lab4

Menu og toolbar laves i Xaml og bør ikke volde de store problemer. Den store udfordring i denne opgave er anvendelsen af Relay commands. Her skal der læses på slide + eksempler. Jeg tager udgangspunkt i eksemplet 11RelayCommandAdv som findes i CommandDemos. Klassen Agents er umiddelbart et godt sted at placere kommandoerne, så jeg flytter Agents klassen ud i sin egen fil og tilføjer properties som returnerer en instans af RelayCommand-klassen, hvor Execute og CanExecute peger på de tilhørende handlere, f.eks.:

```
ICommand _addCommand;
public ICommand AddCommand
{
    get { return _addCommand ?? (_addCommand = new RelayCommand(AddAgent)); }
}

private void AddAgent()
{
    Add(new Agent());
    CurrentIndex = Count - 1;
}
```

Og binder til den i XAML:

```
<MenuItem Header="_Add new" Command="{Binding AddCommand}"/>
<Button ToolTip="Add a new agent" Content="+" Command="{Binding AddCommand}" />
```

Lab5

Tilføjer de klassiske filmenu-valg (new, open, save og saveAs). Det vil være naturligt at bruge de indbyggede fildialoger her, men da dette ikke har været pensum endnu, så bruger jeg en textBox på toolbaren til at angive filnavnet i.

Selve persisteringen af data er efter "bogen" (find de relevante slides). Dog er der en detalje som kan drille. Ved deserialiseringen er man nød til at indsætte agenterne i den eksisterende liste, da det er denne konkrete instans af listen som GUI'en er bindet på. Hvis man sætter agent referencen til at pege på en ny liste vil kontrollerne ikke blive opdateret (men mindre man sætter bindingen på ny).

Bemærk også hvordan jeg i Agents-konstruktoren kun indsætter data i design-mode:

```
    public Agents()
        {
            if ((bool)(DesignerProperties.IsInDesignModeProperty.GetMetadata(
                                        typeof(DependencyObject)).DefaultValue))
            {
                // In Design mode
                Add(new Agent("001", "Nina", "Assassination", "UpperVolta"));
                Add(new Agent("007", "James Bond", "Martinis", "North Korea"));
            }
        }
```

Denne kode indsætter nogle data i Agents klassen når den kaldes i design-mode (Visual Studio laver en instans af `MainWindow` når den viser vinduet i designeren).

Lab6

At tilføje en statusbar få en statusbarItem til at vise antallet ved at binde til count er en let gjort i Xaml. Der er dog et problem: count bliver ikke opdateret automatisk. Jeg er nød til at kalde `NotifyPropertyChanged("Count");` når antallet ændres. Dvs. i Add og Delete command-handlerne.

Uret i højre hjørne laves ved at tilføje en DispatcherTimer og initiere denne i winduets konstruktør. Selve opdateringen sker i timerens tick-eventhandler. Det letteste ville være at skrive direkte til en tekstBlock, men for at vise at man kan binde til forskellige elementer i det samme vindue, så har jeg lavet en Clock-hjælpeklasse som Xaml-koden binder til, og som opdateres fra eventhandleren. Bemærk hvordan denne hjælpeklasse implementerer INotifyPropertyChanged (hvilket er nødvendigt for at få databindingen til at virke). Sådan som jeg har implementeret de øvrige opgaver, så virker GUI'ens databinding, uden at jeg har behøvet at implementere INotifyPropertyChanged for Agentklassen (men det er et særtilfælde – man skal ofte implementere INotifyPropertyChanged for at få databinding til at virke som tilsigtet).