Komunikacja między luźno powiązanymi komponentami

Mechanizm komend

1. Rozpoczynamy od SimplePrismApp z poprzednich zajęć
2. Odnajdujemy Delegate command, który leży w class ButtonViewAViewModel
3. Dołączamy potrzebne biblioteki

```
using Microsoft.Practices.Prism.Mvvm;
using System.Windows.Input;

using System.Windows;
```

4. Używany przez nas delegate command implementuje interfejs ICommand

dzięki temu można zastąpić: na

```
public ICommand SwitchViewCommand { get; set; }
```

5. ICommand wywoływany jest w ButtonViewA.xaml, jak poniżej

```
  <Grid>

<Button Content="Przełącz na widok A" Command="{Binding
SwitchViewCommand}" />

</Grid>
```

6. Przebudujemy to by zastąpić mechanizmem Composite command

Zmieniamy icommand na

```
public ComponentCommand SwitchViewCommand { get; set; }
```

7. Odnajdujemy `SwitchViewCommand = new DelegateCommand(SwitchView);`
8. I zamieniamy na

```
SwitchViewCommand = new CompositeCommand();

SwitchViewCommand.RegisterCommand(new DelegateCommand(SwitchView));
```

9. Uzyskany kod powinien działać, już z wykorzystaniem CompositeCommand

10. Tworzymy nową klasę w ModuleA:

```
public static class GlobalCommands

using Microsoft.Practices.Prism.Mvvm;
using Microsoft.Practices.Prism.Commands;
```

```
public static CompositeCommand SwitchViewCommand = new
CompositeCommand();


GlobalCommands.SwitchViewCommand.RegisterCommand(new
DelegateCommand(SwitchView));
```

11. Przechodzimy do do ButtonAView – powinniśmy znaleźć:

```
<Grid>

   <Button Content="Przełącz na widok A" Command="{Binding
SwitchViewCommand}" />

</Grid>
```

12. Zamieniamy powyższy kod na

```
 <Grid>

    <Button Content="Przełącz na widok A" Command="{x:Static
local:GlobalCommands.SwitchViewCommand}" />

</Grid>
```

13. Koniecznie trzeba pamiętać by dodać:

```
xmlns:local="clr-namespace:ModuleA"
```

14. Uzupełniamy EditorCatalogViewModell.cs, żeby zarejestrować komendę

```
   public ModuleAViewViewModel(IAuthority authority)

        {

             this.authority = authority;

         GlobalCommands.SwitchViewCommand.RegisterCommand(new
DelegateCommand(kominukat));

        }
```

15. Musimy jeszcze napisać funkcję komunikat

```
public void kominukat()

      {

          MessageBox.Show("druga komenda");

      }
```

16. Ewentualne błędy mogą wynikać z braku podłączenia bibliotek:

```
using Microsoft.Practices.Prism.Commands;

using System.Windows;
```

Mechanizm IEventAggregator

1. Dodajemy nową klasę do modułu A, nazywamy ją messageEvent i uzupełniamy:

```
public class messageEvent : CompositePresentationEvent<string> { }
```

2. Dodajemy do EditorCatalogViewModel:

```
IEventAggregator _eventAggregator;
```

3. Do konstruktora dodajemy aggregator

```
        public EditorCatalogViewModel(IProductRepository
productsRepository, IEventAggregator eventAggregator):
base(productsRepository)
```

4. I na początku deklarujemy event aggregator:

```
_eventAggregator = eventAggregator;
```

5. W add product dodajemy:

```
_eventAggregator.GetEvent<messageEvent>().Publish("dodano element");
```

6. W ModuleBViewViewModel dodajemy:

```
IEventAggregator _eventAggregator;
```

7. Aktualizujemy konstruktor:

```
_eventAggregator = eventaggregator;

eventaggregator.GetEvent<messageEvent>().Subscribe(komunikat,true);
```

8. Dopisujemy metodę:

```
public void komunikat(string tekst) { MessageBox.Show(" otrzymano
wiadomość od innego wątku:"+ tekst); }
```

9. Otrzymaliśmy wiadomość przesłaną przez mechanizm EventAggregator