Browsersync: connected

MODULE 20

MODEL-VIEW-VIEWMODEL (MVVM)

MODULE TOPICS

Definition and Evolution of the Pattern
Building Effective ViewModel Objects
Leveraging Bindings and Commands
Minimizing Code-Behind Files
Data Validation and Handling Errors
Building Unit Tests and TDD

MODEL-VIEW-VIEWMODEL (MVVM)

- Approach for creating structurally sound software that is maintainable and understandable
- Helps to cleanly separate the business and presentation logic
- Grows out of Model-View-Presenter (MVP) but diverges in ways that enable you to leverage capabilities of the WPF platform
- Data binding, data templates, commands, behaviors

CATEGORIES OF OBJECTS

- Modelobjects
  - Contain the data consumed and modified by the user
  - Include business rule processing, input validation, change tracking
- View
  - UI control that displays data
  - Allows the user to modify state of the program via device input

VIEWMODEL

- A ViewModel is a model of a view
- Is an abstraction of the user interface
- Should have no knowledge of the UI elements on the screen
  - Logic that deals specifically with objects scoped to a particular viewshould exist in the View's code-behind
- Allows you to treat the UI of an application as a logical system
- Ability to write unit tests for the functionality of the UI
- Views that render ViewModels can be modified or replaced with little or no changes to the ViewModels

CREATING APPLICATIONS

- The fundamental mechanisms in creating applications based on MVVM are data bindingand commands
- ViewModelobjects expose properties to which Views are bound including properties that return command objects
  - The DataContext of a View is set to a ViewModel
  - ViewModeldoes not need a reference to a view
- To expose a modifiable collection, the ViewModelcan use ObservableCollectionfor the View to get change notifications

DEVELOPMENT AND TESTING

- ViewModelclasses are easy to test
  - Views and unit tests are just two different types of ViewModel consumers
  - If you write unit tests for the ViewModelwithout creating any UI objects, you can also completely skin the ViewModel because it has no dependencies on specific visual elements
- Since a view is just an arbitrary consumer of a ViewModel...
  - Development team can focus on creating robust ViewModel classes
  - Design team can focus on creating user-friendly Views

Browsersync: connected

VIEW CLASS

- The view's responsibility is to define the structureand appearanceof what the user sees on the screen
- Ideally, the code-behind of a view contains only a constructor that calls InitializeComponent
  - May contain UI logic that implements visual behavior that is difficult or inefficient to express in XAML
  - Should not contain any logic that you need to unit test
- Views are typically Control-derived or UserControl-derived classes
  - May be represented by a data template

ANY QUESTIONS?