CIS 400: Object-Oriented Design, Development, and Testing

Fall 2023

Exam 2 – 100 points

This test is closed-notes and closed-computers.

There are 10 questions.

Name: ____

Score: ____________

1. (3 pts) What is the purpose of data binding?
    a) To enforce bounds on properties
    b) To prevent data from being changed
    c) To require a data class to implement *INotifyPropertyChanged*
    d) **To synchronize data between a regular C# object and the GUI**

2. (3 pts) What does the ViewModel do in MVVM?
    a) Manages the user interface layout and design
    b) **Applies logic and formatting to the data to create properties that the user interface can bind to**
    c) Tests the user interface by modeling different user interactions
    d) Stores all the data

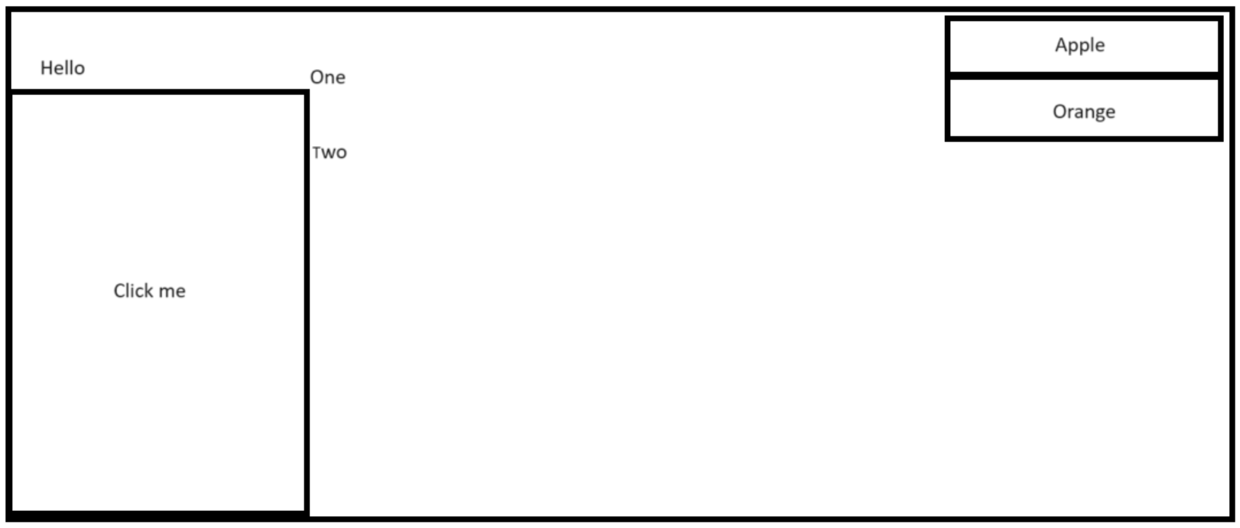
3. (3 pts) Suppose our *MainWindow*'s XAML adds a control named *ExamControl*. Further suppose that in the constructor of *MainWindow*, that we do: "***DataContext = list;***", where *list* has type *List<string>*. If we never set the *DataContext* of *ExamControl*, what value does its *DataContext* have?
    a) ***ExamControl*'s *DataContext* is also *list***
    b) *ExamControl*'s *DataContext* is null
    c) *ExamControl*'s *DataContext* is the *MainWindow*
    d) We will have a compilation error because *List<string>* is not a valid *DataContext*

4. (3 pts) Consider the *ButtonNameEventArgs* and *SampleControl* classes in the handout at the end of the exam. Within *SampleControl*, how would we declare an event named *ButtonEvent* that used *ButtonNameEventArgs* as our custom event args?
    a) ***public ButtonEvent<ButtonNameEventArgs>;***
    b) ***public event<ButtonNameEventArgs> ButtonEvent;***
    c) ***public event EventHandler<ButtonNameEventArgs> ButtonEvent;***
    d) ***public EventHandler ButtonEvent<ButtonNameEventArgs>;***

5. (3 pts) Assume *ButtonEvent* from #4 has been defined in *SampleControl*. Suppose the *ButtonClick* method is the Click event handler for several buttons defined in the *SampleControl*'s XAML, and that the *Name* property for each button has been set. As if we were inside the *ButtonClick* method, how would we invoke *ButtonEvent* to include the clicked button's name? (You may want to refer to the handout.)
   a) ***ButtonEvent?.Invoke(this, new ButtonNameEventArgs(sender.Name));***
   b) ***new ButtonEvent?.Invoke(new ButtonEventArgs(sender));***
   c) ***ButtonEvent?.Invoke(this, new ButtonNameEventArgs((sender as Button).Name));***
   d) ***new ButtonEventArgs?.Invoke(ButtonEvent);***

6. (3 pts) Following the work in #4-5 and referring to the handout, suppose that *MainWindow* adds a *SampleControl* named *MySample* in its XAML. If we wanted *MainWindow* to be able to do something whenever *ButtonEvent* was invoked, what would need to do?
   a) **Create an event handler in *MainWindow* and attach that event handler to *MySample.ButtonEvent***
   b) Make *SampleControl* implement *INotifyPropertyChanged*
   c) We can't access *MainWindow* from *SampleControl* without adding it as a field
   d) Create an event handler in *MainWindow* and attach that event handler to *SampleControl.ButtonEvent*

7. (22 pts) Draw the *DrawXAML* control defined in the handout as it will appear on screen:



8. (30 pts) Write the class *Rectangle* as shown in the UML from the handout. Be sure that its *Area* and *Perimeter* properties correctly give the area and perimeter of a rectangle, that the result of *ToString()* has the format “3x4 rectangle, area 12, perimeter 14”, and that the class correctly implements the *INotifyPropertyChanged* interface.

```
public class Rectangle : INotifyPropertyChanged {
   public event PropertyChangedEventHandler? PropertyChanged;

   public Rectangle(int len, int wid) {
         _length = len;
         _width = wid;
   }

   private int _length;
   public int Length {
         get => _length;
         set {
               _length = value;
               PropertyChanged?.Invoke(this,
                     new PropertyChangedEventArgs(nameof(Length)));
               PropertyChanged?.Invoke(this,
                     new PropertyChangedEventArgs(nameof(Area)));
               PropertyChanged?.Invoke(this,
                     new PropertyChangedEventArgs(nameof(Perimeter)));
         }
   }

   private int _width;
   public int Width {
         get => _width;
         set {
               _width = value;
               PropertyChanged?.Invoke(this,
                     new PropertyChangedEventArgs(nameof(Width)));
               PropertyChanged?.Invoke(this,
                     new PropertyChangedEventArgs(nameof(Area)));
               PropertyChanged?.Invoke(this,
                     new PropertyChangedEventArgs(nameof(Perimeter)));
         }
   }

   public int Area => Length*Width;
   public int Perimeter => 2*Length + 2*Width;

   public override string ToString() {
         return $"{Length} x {Width} rectangle, area {Area}, perimeter
{Perimeter}";
```

```
        }
    }
```

9. (15 pts) Complete the XAML below to display the *Length*, *Width*, *Area*, and *Perimeter* properties from the *Rectangle* class (use whatever controls you want as long as they display the appropriate values):

```
<StackPanel>
      <TextBlock Text="{Binding Path=Length}"/>
      <TextBlock Text="{Binding Path=Width}"/>
      <TextBlock Text="{Binding Path=Area}"/>
      <TextBlock Text="{Binding Path=Perimeter}"/>

</StackPanel>
```

10. (15 pts) Complete both the inline data and the implementation for the following unit test of the *Rectangle* class:

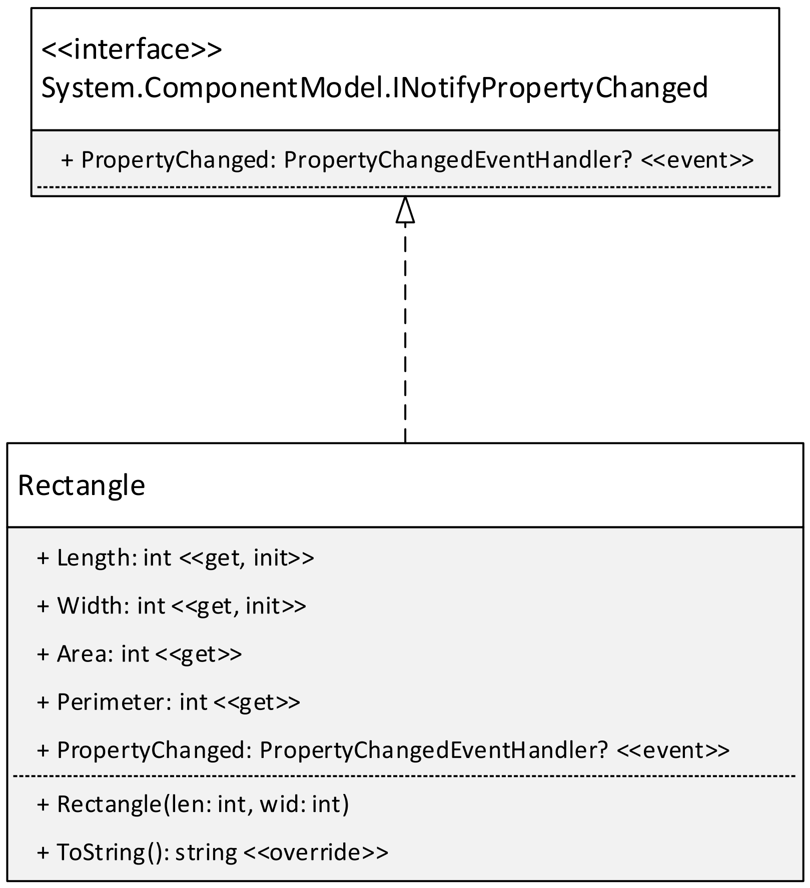
```
[Theory]
[InlineData(2, "Width")]
[InlineData(3, "Area")]
[InlineData(2, "Perimeater")]
public void ChangingWidthShouldNotifyOfPropertyChange(int width,
                        string propertyName)
{
      Rectangle r = new(1, 1);
      Assert.PropertyChanged(r, propertyName, () => {
            r.Width = width;
      });
}
```

HANDOUT - Feel free to remove this portion to make it easier to work.

```
//This page is needed for the multiple choice problems

public class ButtonNameEventArgs : RoutedEventArgs
{
      public string ButtonName { get; init; }

      public ButtonNameEventArgs(string n)
      {
            ButtonName = n;
      }
}

public class SampleControl : UserControl
{
      //the constructor is hidden (and not relevant)

      private void ButtonClick(object? sender, RoutedEventArgs e)
      {

      }
}
```

```
//This page is needed for #7

<UserControl x:Class="Exam2.DrawXAML" ...
      d:DesignHeight="450" d:DesignWidth="800">

      <Grid>
            <Grid.ColumnDefinitions>
                  <ColumnDefinition Width="1*"/>
                  <ColumnDefinition Width="3*"/>
                  <ColumnDefinition Width="1*"/>
            </Grid.ColumnDefinitions>
            <Grid.RowDefinitions>
                  <RowDefinition Height="1*"/>
            </Grid.RowDefinitions>

            <DockPanel Grid.Row="0" Grid.Column="0">
                  <TextBlock Text="Hello" DockPanel.Dock="Top"/>
                  <Button DockPanel.Dock="Bottom">Click me</Button
            </DockPanel>
            <Grid Grid.Row="0" Grid.Column="1">
                  <Grid.RowDefinitions>
                        <RowDefinition Height="1*"/>
                        <RowDefinition Height="1*"/>
                        <RowDefinition Height="3*"/>
                  </Grid.RowDefinitions>
                  <TextBlock Text="One" Grid.Row="0"/>
                  <TextBlock Text="Two" Grid.Row="1"/>
            </Grid>
            <StackPanel Grid.Row="0" Grid.Column="2">
                  <Button>Apple</Button>
                  <Button>Orange</Button>
            </StackPanel>
      </Grid>

</UserControl>
```

```
//This page is needed for #8-10
```



<<interface>>
System.ComponentModel.INotifyPropertyChanged
+ PropertyChanged: PropertyChangedEventHandler? <<event>>
Rectangle
+ Length: int <<get, init>>
+ Width: int <<get, init>>
+ Area: int <<get>>
+ Perimeter: int <<get>>
+ PropertyChanged: PropertyChangedEventHandler? <<event>>
+ Rectangle(len: int, wid: int)
+ ToString(): string <<override>>