This document contains review problems for Exam 2 in CIS 400.

1. What is the purpose of data binding?

Allows us to share information between the front end (GUI) and the back end (data) – changing in one place changes in both

2. What is the purpose of the *INotifyPropertyChanged* interface?

If a class implements INotifyPropertyChanged, we know that it has a PropertyChanged event that (hopefully) is being invoked whenever something happens that cause the value of a property to change. We then know we can attach an event handler to an object's PropertyChanged event to have something happen when the object's properties change. This happens a lot with GUI development – when we bind to a property whose class implements INotifyPropertyChanged, code is automatically generated that attaches an event handler to the PropertyChanged event. That event handler re-renders the control, which ensures that the control always displays the current values of properties.

3. How can we send a message from Control1 to Control2? Explain in detail.

With a custom event.

1) Create a custom event args class whose property is the info we want to send in our message (CustomEventArgs)

2) Define the event in Control1

```
public event EventHandler<CustomEventArgs>? eventName;
```

3) whenever we want to send the message, in Control1 do:

```
eventName?.Invoke(this, new CustomEventArgs(info));
```

3) in Control2, subscribe to the event (assume control1objectName is the name of the Control1 object as it was placed on the Control2 XAML):

```
control1objectName.eventName += MyEventHandler;
```

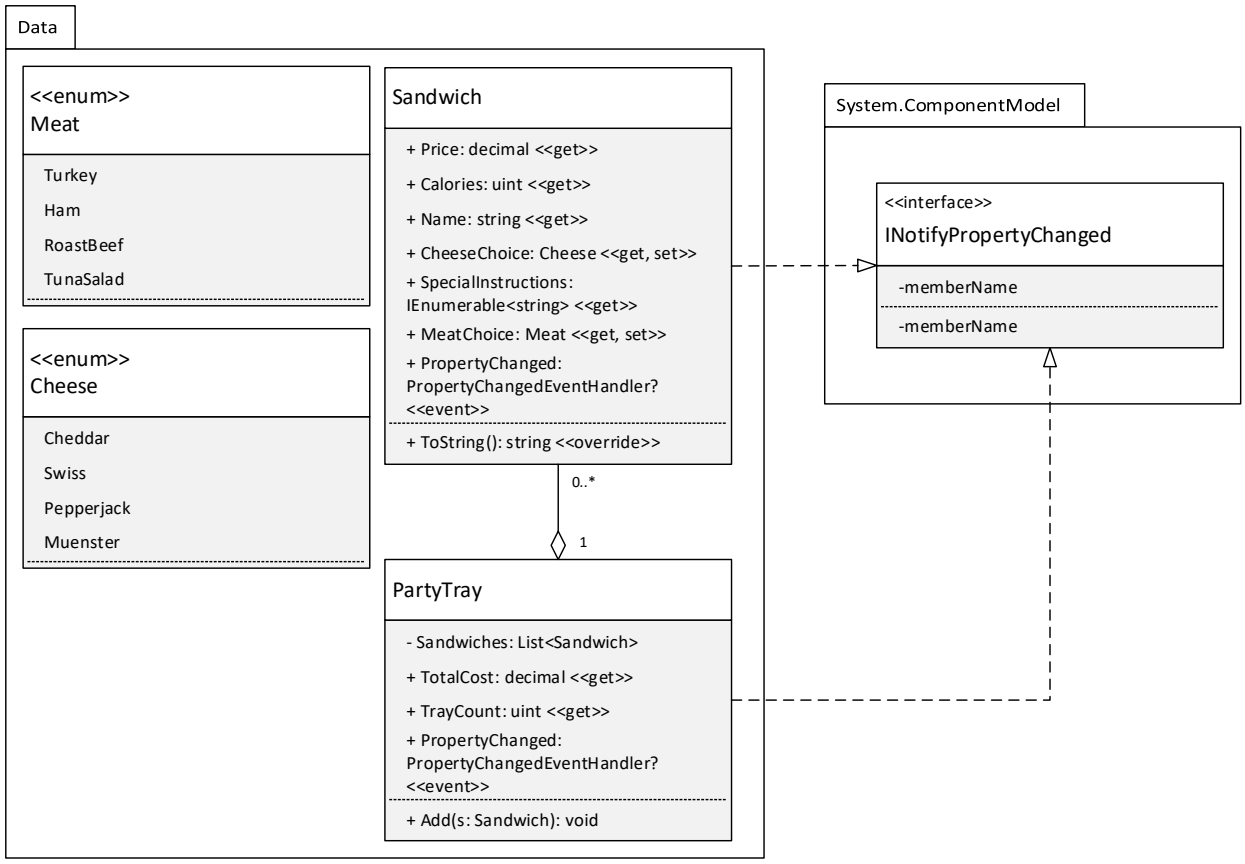
4) In Control2, write the event handler:

```
public void MyEventHandler(object? sender, CustomEventArgs e) {
    //e.Info (or whatever property) to get the info from the event
}
```

4. Implement the *Sandwich* class by referring to the UML below, with the exception of the *Price*, *Calories*, *Name*, and *SpecialInstructions* properties. (You may assume that changing the *CheeseChoice* and *MeatChoice* affect the values of the *Price*, *Calories*, and *SpecialInstructions*.) Be sure to provide functionality for implementing *INotifyPropertyChanged*.

```
public class Sandwich : INotifyPropertyChanged {
	public event PropertyChangeEventHandler? PropertyChanged;
	private Cheese _cheese;
	public Cheese CheeseChoice {
		get => _cheese;
		set {
			_cheese = value;
			PropertyChanged?.Invoke(this,
				new PropertyChangedEventArgs(nameof(Price)));
			PropertyChanged?.Invoke(this,
				new PropertyChangedEventArgs(nameof(Calories)));
			PropertyChanged?.Invoke(this,
				new PropertyChangedEventArgs(nameof(SpecialInstructions)));
			PropertyChanged?.Invoke(this,
				new PropertyChangedEventArgs(nameof(CheeseChoice)));
		}
	}
	private Meat _meat;
	public Meat MeatChoice {
		get => _meat;
		set {
			_cheese = value;
			PropertyChanged?.Invoke(this,
				new PropertyChangedEventArgs(nameof(Price)));
			PropertyChanged?.Invoke(this,
				new PropertyChangedEventArgs(nameof(Calories)));
			PropertyChanged?.Invoke(this,
				new PropertyChangedEventArgs(nameof(SpecialInstructions)));
			PropertyChanged?.Invoke(this,
				new PropertyChangedEventArgs(nameof(MeatChoice)));
		}
	}

	public override string ToString() {
		return Name;
	}
}
```



5. Suppose the *PartyTray* class is implemented as shown in the UML diagram from #4, except for the functionality for the *INotifyPropertyChanged* implementation. Explain what would need to be done to ensure that *PartyTray*'s *PropertyChanged* event was invoked to say its *TotalCost* had changed any time one of its sandwiches was customized (i.e., whenever a property changed for a sandwich).

1) Write a method in PartyTray that we want to execute when a sandwich is customized

```
private void OnSandwichChange(object? sender, PropertyChangedEventArgs e){
        //this is PartyTray's PropertyChanged
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(nameof(TotalCost)));
}
```

In PartyTray's Add method, attach our event handler to the new sandwich:

```
        s.PropertyChanged += OnSandwichChange;
```

6. Write a unit test to assert that the *PropertyChanged* event for *Sandwich* is invoked for the *Calories* property whenever *CheeseChoice* is changed.

```
[Theory]
[InlineData(Cheese.Swiss)]
//and similarly inline data for all cheeses in the enum
public void CheeseChangeInvokesCalorieChange(Cheese cheese) {
        Sandwich s = new();
        Assert.PropertyChanged(s, "Calories", () => {
                s.CheeseChoice = cheese;
        });
}
```

7. Suppose we have a control to display customizations for our *Sandwich*. What does the control look like when it is displayed?

```
<UserControl x:Class="Views.CustomizeSandwich"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:data="clr-namespace:Data;assembly=data"
        xmlns:system="clr-namespace:System;assembly=mscorlib"
        xmlns:local="clr-namespace:Views"
        mc:Ignorable="d"
        d:DesignHeight="450" d:DesignWidth="800">
```

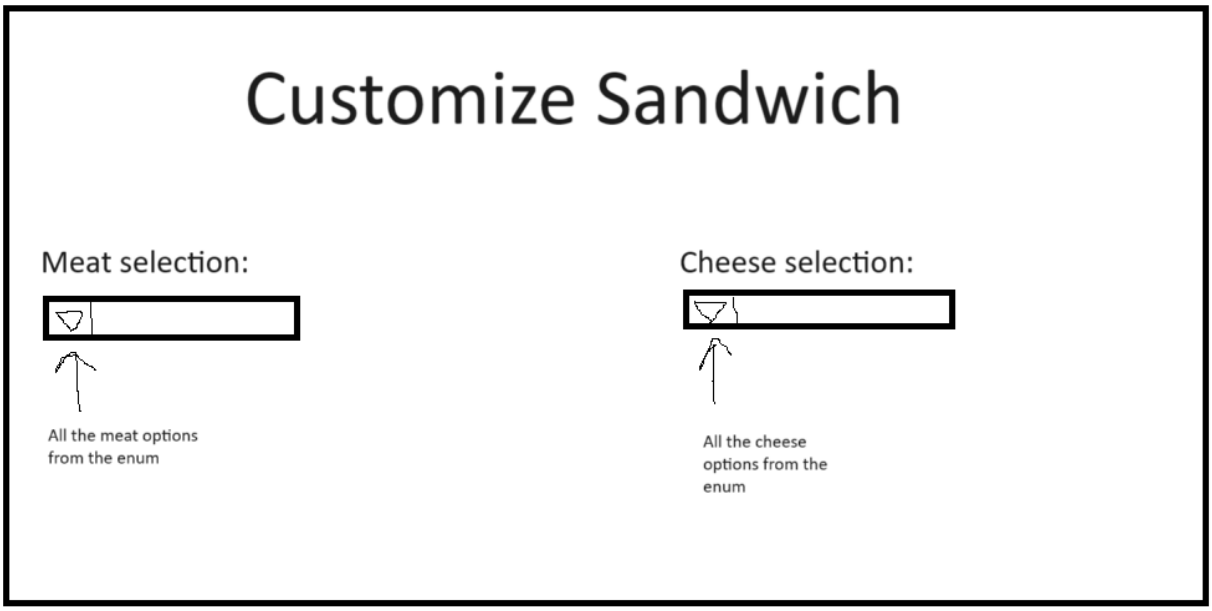
```
<UserControl.Resources>

        <ObjectDataProvider x:Key="meat" MethodName="GetValues" ObjectType="{x:Type
        system:Enum}">
                <ObjectDataProvider.MethodParameters>
                        <x:Type TypeName="data:Meat"/>
                </ObjectDataProvider.MethodParameters>
        </ObjectDataProvider>

        <ObjectDataProvider x:Key="cheese" MethodName="GetValues" ObjectType="{x:Type
        system:Enum}">
                <ObjectDataProvider.MethodParameters>
                        <x:Type TypeName="data:Cheese"/>
                </ObjectDataProvider.MethodParameters>
        </ObjectDataProvider>
</UserControl.Resources>

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

        <TextBlock Text="Customize Sandwich" FontSize="48" Grid.Row="0" Grid.ColumnSpan="2"
                TextAlignment="Center"/>
        <StackPanel Grid.Column="0" Grid.Row="1">
                <TextBlock Text="Meat selection:" FontSize="20"/>
                <ComboBox ItemsSource="{Binding Source={StaticResource meat}}"/>
        </StackPanel>
        <StackPanel Grid.Column="1" Grid.Row="1">
                <TextBlock Text="Cheese selection: " FontSize="20"/>
                <ComboBox ItemsSource="{Binding Source={StaticResource cheese}}"/>
        </StackPanel>
</Grid>
</UserControl>
```



8. What would we need to do with our XAML to bind the SelectedItem in each ComboBox to the associated properties in Sandwich? What would need to happen for our CustomizeSandwich control to then display the information for a particular sandwich?

Need to make the DataContext for that CustomizeSandwich control be a Sandwich object

In the meats combo box, do:

SelectedItem="{Binding Path=MeatChoice}"

In the cheese combo box:

SelectedItem="{Binding Path=CheeseChoice}"

If changing something in the data would impact these properties, would need Sandwich to implement INotifyPropertyChanged and invoke PropertyChanged when the properties would be affected (which was already done in a previous problem).