МОДЕЛЬ КЛАССОВ ФРЕЙМВОРКА WPF

Фреймворк представляет из себя набор классов и методов, необходимых для оформления графических приложений.

Единицей для построения графического интерфейса является элемент. ***Элемент*** - это такой объект, который может участвовать в компоновке и реагировать на действия. Элементы так же могут быть классифицированы по назначению: элементы управления содержимым, элементы отображения и т.п.

Для того, чтобы понять, какие классы составляют элементы, необходимо разобраться с задачами, которые возложены на фреймворк.

1) ***Управление потоком***. В `WPF` принято решение, что отрисовка происходит в одном синхронном потоке. (Поток - это единица выполнения).

2) ***Управление рисованием элемента***. Отрисовка элемента происходит исходя из функций, предоставляемых низкоуровневыми графическими библиотеками. *(WPF использует DirectX)*.

3) ***Управление событиями элемента***. Любой элемент может порождать события. Для фреймворка события должны быть унифицированы, чтобы была возможность управлять ими на более низком уровне.

Исходя из этих трёх задач принята следующая модель, которую вы можете найти в документации к фреймворку.

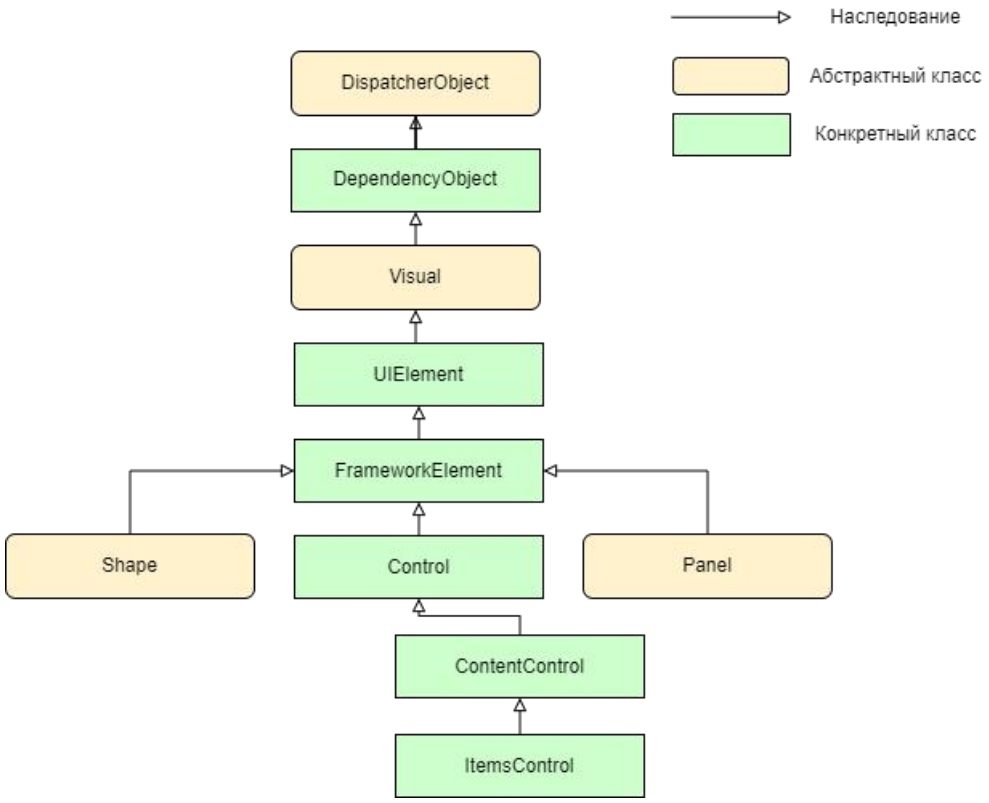

Рисунок 1 - модель классов WPF

Абстрактный класс является шаблоном для построения других элементов, представляя набор базовых методов и других составляющих.

Конкретный класс может иметь экземпляр и уже реализует тот функционал, который был представлен абстрактными классами. И как можно помнить, в **C#** имеется возможность множественного наследования. Таким образом, каждый класс является как бы модулем. А каждый класс может содержать несколько модулей. Множество классов из этой модели решает какую-либо конкретную область задач.

Рассмотрим модель в иерархическом порядке - сверху вниз.

DispatcherObject

Начинает иерархию класс, который управляет элементом в потоке выполнения приложения **WPF**. В **WPF** используют однопоточную модель **STA** *(single-thread affinity)*. На каждое окно выделяется один поток. Как можно знать, программа, выполняемая в одном потоке выполняет последовательность команд, ожидая выполнения каждой команды. Это значит, что все элементы отрисовываются в определённом порядке.

Класс **`DispatcherObject`** существует у каждого элемента и предоставляет функции доступа к потоку отрисовки.

Дело в том, что поток отрисовки является защищённым и из других потоков *(например, созданных с помощью класса **`Thread`**)* получать доступ нельзя. Диспетчер проверяет поток, который пытается получить доступ к элементу на принадлежность к этому потоку (как можно знать, у каждого потока есть уникальный идентификатор). Если ID потока разные - вызывается исключение.

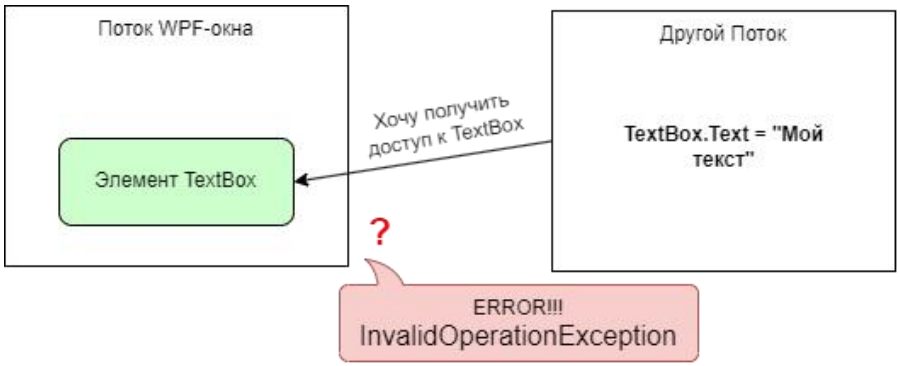

Рисунок 2

Для управления элементом из другого потока необходимо «получить разрешение» у его диспетчера. Для этого, **`DispatcherObject`** реализует свойство **`Dispatcher`** для каждого элемента WPF.

Именно поэтому этот класс начинает иерархию модели.

DependencyObject

Этот класс управляет свойствами, значения которых зависят от других свойств. **`DependencyObject`** предоставляет возможность элементу использовать множество **`DependencyProperty`** - конкретные свойства зависимости. Можете воспринимать это свойство как разъём, к которому извне объекта можно подключиться и положить значение.

Так же, **`DependencyObject`** может реагировать на свойства, которые «прикрепляются» к нему во время выполнения программы.

Со свойствами зависимости разработчик **`WPF`** будет иметь дело постоянно.

Рассмотрим пример. Элемент Button имеет множество свойств. От значений этих свойств зависит отображение этого элемента. То есть, **Button** зависит от свойств.

```
<Button Height="100" Width="100" Content="Кнопка"></Button>
```

Рисунок 3

Свойства зависимости так же предоставляют возможность отслеживания изменений во время выполнения программы. Это очень удобный инструмент, которым разработчик будет пользоваться часто.

Visual

До этого класса, другие классы управляли целостностью потока выполнения и данными элемента. После того, как все данные для элемента были установлены, **DirectX** получает все необходимые данные для того, чтобы отрисовать элемент.

Класс Visual отвечает за отрисовку элемента. Нужно учесть, что это абстрактный класс, поэтому просто элемент Visual создать нельзя. Этот класс просто предоставляет функционал по отрисовке, в частности функцию **OnRender()**, которая из графических примитивов отображает элемент. Поэтому для этого класса нужен конкретный класс. Как написано в книге Мак-Дональда, если вы просто хотите отрисовывать элемент вручную без WPF, то можете пользоваться этим классом.

UIElement

Уже предоставляет конкретный элемент Visual. В классе **UIElement** определены функции компоновки, ввода, фокуса и события. Все щелчки мышью, ввода с клавиатуры, зависимости компоновки оформлены в виде событий. Так же, событиями предоставляется возможность управлять с помощью удобного интерфейса команд.

FrameworkElement

Класс **FrameworkElement** является конечным пунктом в дереве наследования. Он оборачивает функционал **UIElement**, описывая свойства и методы для взаимодействия с ним.

Этот класс является базовым для построения пользовательского элемента «с нуля». Этот класс как чистый лист для `WPF`, у которого есть свойства «встраивания» в интерфейс: выравнивание элемента, отступы и т.п.

Грубо говоря, элемент, наследуемый от `FrameworkElement`, уже воспринимается WPF как элемент, который может участвовать в компоновке.

Shape

Этот класс представляется основой для базовых фигур, например, `Rectangle`, `Polygon`, `Ellipse`, `Line`, `Path`. Эти фигуры так же могут являться основой более сложных элементов, например, `TextBox`.

Control

Какая первая ассоциация у вас возникает со словом «контроллер»? наверное, это элемент, который может контролировать что-то. Кнопка контролирует нажатие, поле ввода управляет вводимым текстом.

Так же мы знаем, что контроллером можно управлять, ведь кнопка нажимается пользователем, текст вводится тоже пользователем...

Для подобных контроллеров есть общий класс - `Controller`. Он представляет элемент, который управляется пользователем. Это динамичный элемент интерфейса. Все контроллеры имеют набор событий, которые могут быть вызваны пользователем, например, `ButtonClick`.

Так же, для контроллеров немаловажна поддержка модификации отображения: ширина, высота, цвет, содержимое и прочие свойства могут быть переопределены.

Класс `Control` так же предоставляет мощный инструмент поддержки шаблонов, которые можно использовать для переопределения стандартного вида элемента. Шаблоны можно использовать для создания собственных стилей.

Элементы, которые наследуются от Control называются элементами управления.

ContentControl

Этот класс предназначен для управления содержимым. Т.е. Элемент представляет из себя некоторый контейнер, в котором могут находиться элементы, которыми он может управлять. Например, элемент `Window`. Примечательно то, что содержимым `ContentControl` может быть набор любых элементов - строки, фигуры и т.п.

ItemsControl

`ItemsControl` предоставляет элемент, для управления набором каких-либо единиц информации. По сути это элемент списка. Каждая единица списка и называется `Item` - записью. `ListBox`, `TreeView` и подобные элементы являются наследниками `ItemsControl`.

Panel

Этот класс является базовым для элементов компоновки других элементов. Их ещё называют контейнерами. Он предоставляет методы по упорядочиванию элементов и установки их размеров. Класс `Panel` - ключевой механизм динамичного интерфейса `WPF`.