1. Перечислите преимущества и недостатки WPF?

серьёзным преимуществом я бы назвал понятие привязки (Binding) и DataContext, которые радикально облегчают написание правильно структурированных программ, в которых представление отделено от модели, бизнес-логики и контента.

1. поддерживает Windows Forms
2. проще анимация и специальные эффекты
3. в возможность повторного использования существующего кода
4. легко создать собственный внешний вид
5. привязка данных, использование таких элементов, как стили, шаблоны, темы и др.

Недостатки WPF:

- Высокая стоимость разработки приложений;

- Для использования WPF необходима установка .NET Framework;

- Нередко имеет более высокие требования к аппаратному обеспечению.

2. Зачем нужен язык XAML? Каким образом он используется в WPF?

XAML (eXtensible Application Markup Language) - язык разметки, используемый для описания пользовательского интерфейса в WPF. XAML позволяет разработчику создавать графические элементы и интерфейсы с помощью текстового описания вместо программирования. В XAML описываются визуальные элементы, свойства и события элементов управления, а также привязки данных.

3. Какие бывают контейнеры компоновки?

- Canvas - позволяет позиционировать элементы управления вручную на холсте;

- Grid - разбивает пользовательский интерфейс на таблицу с ячейками для размещения элементов управления;

- UniformGrid - Аналогичен контейнеру Grid, но все столбцы и строки одинакового размера

- StackPanel - выравнивает элементы управления в указанной ориентации, добавляя их в стек;

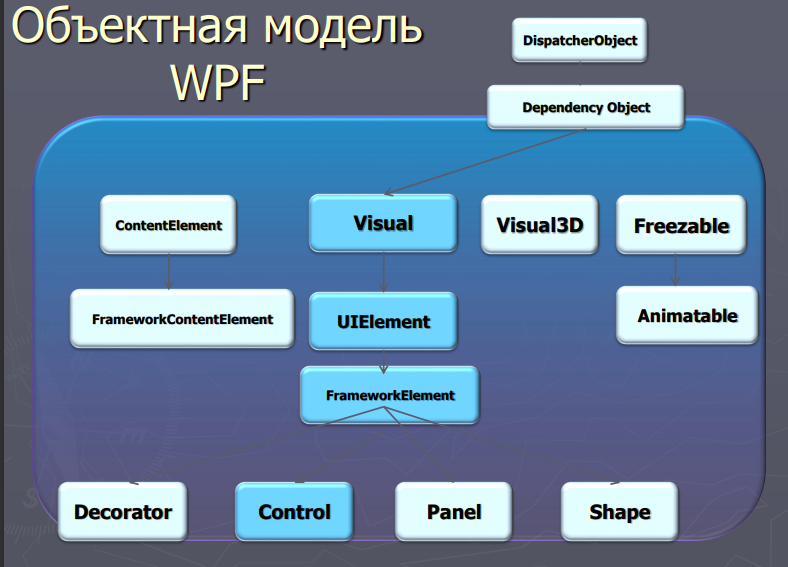
- WrapPanel - выравнивает элементы управления в указанной ориентации, перенося их на другую строку при необходимости;

- DockPanel - размещает элементы управления в заранее определенных зонах (верхняя, нижняя, левая, правая, центральная).

- Viewbox- Элемент (графика) может самостоятельно подгонять свои размеры к окну

4. Перечислите основные группы элементов управления. Расскажите про объектную модель WPF.

* **Элементы управления содержимым**, например кнопки (Button), метки (Label)
* **Специальные контейнеры**, которые содержат другие элементы, но в отличие от элементов Grid или Canvas не являются контейнерами компоновки - ScrollViewer,GroupBox
* **Декораторы**, чье предназначение создание определенного фона вокруг вложенных элементов, например, Border или Viewbox.
* **Элементы управления списками**, например, ListBox, ComboBox.
* **Текстовые элементы управления**, например, TextBox, RichTextBox.
* **Элементы, основанные на диапазонах значений**, например, ProgressBar, Slider.
* **Элементы для работ с датами**, например, DatePicker и Calendar.
* **Остальные элементы управления**, которые не вошли в предыдущие подгруппы, например, Image.



Объектная модель WPF делятся на следующие элементы:

1. FrameworkElement - это базовый класс для всех элементов пользовательского интерфейса в WPF. Элементы, производные от него, могут иметь дочерние элементы, свойства, события и методы, определенные в этом классе.

2. Control - это класс, который расширяет FrameworkElement, предоставляя дополнительные возможности для создания элементов управления (например, кнопки, текстовые поля и т.д.). Класс Control включает в себя свойства, события и методы, которые позволяют настраивать и изменять внешний вид элемента управления, его поведение и т.д.

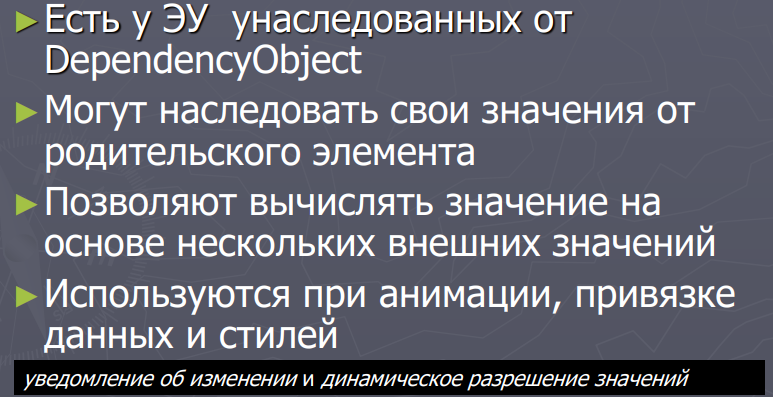
3. UIElement - это класс, который определяет основные элементы пользовательского интерфейса, такие как кнопки, текстовые поля, списки, изображения и т.д. Он включает в себя свойства, события и методы, которые позволяют настраивать отображение и поведение элемента.

4. Visual - это класс, который определяет графические объекты, используемые для отрисовки элементов пользовательского интерфейса. Он включает в себя методы, которые позволяют рисовать и изменять графические объекты, такие как прямоугольники, линии и т.д.

5. Объясните назначение класса DependencyProperty. Зачем нужны свойства зависимостей? Как создать новой свойство зависимости и в каких случаях это необходимо?

Без свойств зависимостей были бы невозможны многие ключевые особенности WPF, как привязка данных, стили, анимация и т.д.

Для того чтобы в классе можно было реализовывать Dependency Property необходимо, чтобы класс наследовался от DependencyObject либо его наследников



Представляет свойство, которое можно задать с помощью методов, например стили, привязки данных, анимации и наследование.

// свойство зависимостей

public static readonly DependencyProperty TextProperty;

// Регистрация свойства

TextProperty = DependencyProperty.Register(

"Text",

typeof(string),

typeof(TextBlock),

new FrameworkPropertyMetadata(

string.Empty,

FrameworkPropertyMetadataOptions.AffectsMeasure |

FrameworkPropertyMetadataOptions.AffectsRender,

new PropertyChangedCallback(OnTextChanged),

new CoerceValueCallback(CoerceText)));

// Обычное свойство .NET - обертка над свойством зависимостей

public string Text

{

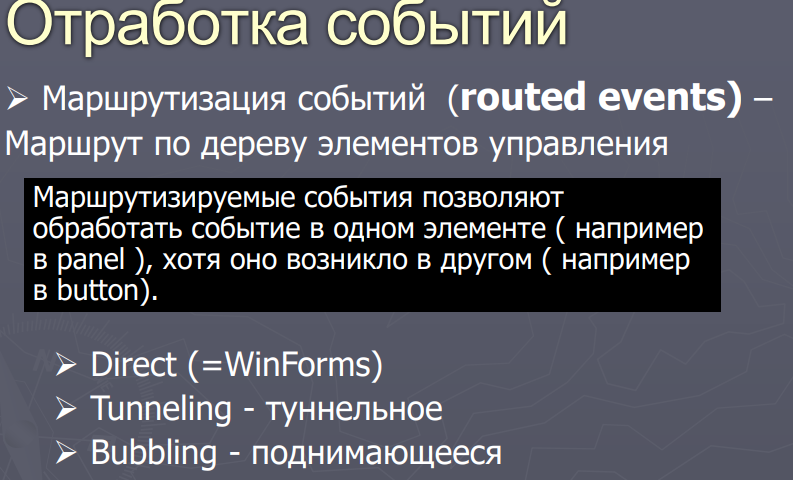
get { return (string) GetValue(TextProperty); }

set { SetValue(TextProperty, value); }

}

// метод, вызываемый при изменении значения свойства

6. Каким образом осуществляется обработка событий в WPF? Что означает концепция маршрутизированных событий? Перечислите основные группы событий.



Обработка событий в WPF осуществляется с помощью механизма обработки событий, который основан на делегатах (event delegates) и маршрутизации событий (event routing).

**Концепция маршрутизированных событий в WPF** (Windows Presentation Foundation) - это механизм событий, который позволяет элементам WPF гибко обрабатывать и передавать события в разных направлениях, в зависимости от их местоположения в иерархии элементов.

Так, маршрутизируемые события делятся на три вида:

* **Прямые** (direct events) - они возникают и отрабытывают на одном элементе и никуда дальше не передаются. Действуют как обычные события.
* **Поднимающиеся** (bubbling events) - возникают на одном элементе, а потом передаются дальше к родителю - элементу-контейнеру и далее, пока не достигнет наивысшего родителя в дереве элементов.
* **Опускающиеся, туннельные** (tunneling events) - начинает отрабатывать в корневом элементе окна приложения и идет далее по вложенным элементам, пока не достигнет элемента, вызвавшего это событие.
* **Прикрепляемые события** (Attached events) - Если у нас есть несколько элементов одного и того же типа и мы хотим привязать их к одному событию, то мы можем воспользоваться прикрепляемыми событиями.

Определение маршрутизированных событий отличается от стандартного определения событий в языке C#. Для определения маршрутизированных событий в классе создвалось статическое поле по типу RoutedEvent:

1 public static RoutedEvent СобытиеEvent

Затем это событие регистрируется в статическом конструкторе.

Для элементов управления в WPF определено большое количество событий, которые условно можно разделить на несколько групп:

* События клавиатуры
* События мыши
* События стилуса
* События сенсорного экрана/мультитач
* События жизненного цикла

7. Что такое Resource Dictionary?

Ресурсный словарь - это независимая коллекция ресурсов в отдельном файле XAML, которую можно импортировать в коллекцию Resources любого элемента приложения WPF.

Предназначен для хранения различных ресурсов приложения, таких как стили, шаблоны, кисти, пиктограммы, изображения, шрифты и т.д.

8. Что такое стиль и как его создать? В чем преимущество использования стиля?

В WPF стиль (style) представляет собой набор свойств, применяемых к элементу управления, определенному разработчиком. Стиль позволяет задать визуальное оформление элементов управления, таких как кнопки, текстовые поля, список и т.д., а также изменять их поведение.

Стиль в WPF можно создать с помощью тега <Style> в разметке XAML. В стиле можно указать свойства, такие как шрифт, цвет, размер и т.д., а также события и привязки данных. Стиль может быть определен внутри элемента управления или в ресурсах приложения, что позволяет повторно использовать его в различных частях приложения.

9. Что такое Command? Расскажите паттерн Command. Как в WPF используется Command и для чего?

Command (команда) в WPF - это логика, определяющая, что должно происходить при выполнении действия пользователем (например, нажатие кнопки) на элементе управления. Command - это реализация паттерна команда (Command Pattern) в WPF, который позволяет создавать объекты команд и использовать их для абстрагирования логики приложения от элементов управления.

Паттерн Command используется для достижения следующих целей:

- разделение логики выполнения команд от элементов управления;

- переиспользование команд для разных элементов управления;

- поддержка отмены и повтора выполнения команд;

- поддержка горячих клавиш и меню.

В WPF Command используется для связывания действий пользователя (например, нажатие кнопки) с определенным методом в коде приложения. При использовании Command логика действия пользователя вынесена из элемента управления и перенесена в класс команды, что делает приложение гибче и удобнее в плане сопровождения.

Чтобы создать Command в WPF, необходимо создать класс, реализующий интерфейс ICommand, который включает в себя методы для выполнения команды и проверки ее доступности. Затем нужно создать свойство зависимости типа ICommand внутри элемента управления и связать его с созданным классом команды в коде приложения.

Например, чтобы реализовать команду "Открыть файл", нужно создать класс OpenFileCommand, который будет реализовывать ICommand. В этом классе нужно определить два метода: Execute (выполнение команды) и CanExecute (возможность выполнения команды). Затем нужно создать свойство зависимости типа ICommand в элементе управления, например, MenuItem, и связать его с созданным классом команды с помощью Binding.

Команда может быть использована для связывания многих элементов управления, например, кнопок, меню, контекстных меню и других элементов, с одним и тем же методом в коде приложения. Это позволяет упростить код и изменять логику приложения без изменения элементов управления.

Command — это команда, которая будет выполняться при вызове источника команды. CommandTarget — это объект, для которого выполняется команда. Следует отметить, что в WPF свойство CommandTarget для ICommandSource применимо, только когда ICommand — RoutedCommand.

Паттерн "Команда" (Command) позволяет инкапсулировать запрос на выполнение определенного действия в виде отдельного объекта. Этот объект запроса на действие и называется командой. При этом объекты, инициирующие запросы на выполнение действия, отделяются от объектов, которые выполняют это действие.

Команды представляют механизм выполнения какой-нибудь задачи, например, копирования текста - когда мы нажимаем Ctrl+C, то мы копируем текст в буффер. В процессе копирования выполняется ряд действий, и все вместе эти действия объединяются в одну команду. ... Функция CanExecute возвращает true, если команда включена и доступна для использования, и false, если команда отключена.