1. **Что такое Свойства зависимости? Для чего они нужны**

В приложениях WPF свойство зависимости – это особый тип свойства, который расширяет свойство CLR. класс Button поддерживает **свойство** зависимостей **IsMouseOver** .

Свойство Dependency дает вам все преимущества, когда вы используете его в своем приложении. Свойство зависимости можно использовать над свойством CLR в следующих сценариях:

* Если вы хотите установить стиль
* Если вы хотите привязку данных
* Если вы хотите установить с ресурсом (статический или динамический ресурс)
* Если вы хотите поддержать анимацию

По сути, свойства зависимости предлагают множество функций, которые вы не получите, используя свойство CLR.

Основное различие между **свойствами зависимости** и другими **свойствами CLR**

* Свойства CLR могут напрямую читать / писать из закрытого члена класса, используя **getter** и **setter** . В отличие от свойств зависимости не хранятся в локальном объекте.
* Свойства зависимостей хранятся в словаре пар ключ / значение, который предоставляется классом DependencyObject. Это также экономит много памяти, поскольку сохраняет свойства при изменении. Это также может быть связано в XAML.

**2 Как создать DependencyProperty**

Объявите и зарегистрируйте **свойство зависимости** с помощью регистра системных вызовов.

Предоставить **установщик** и **получатель** для свойства.

Определите **статический обработчик,** который будет обрабатывать любые изменения, которые происходят глобально

Определите **обработчик экземпляра,** который будет обрабатывать любые изменения, которые происходят с этим конкретным экземпляром.

**3. Для чего и как используют делегат ValidateValueCallback?**

**4. Для чего и как используют делегат CoerceValueCallback?**

1) ValidateValueCallback: делегат - true и false – прошло или нет проверку – срабатывает первым

2) CoerceValueCallback: делегат, который может подкорректировать уже существующее значение свойства, если оно вдруг не попадает в диапазон допустимых значени срабатывает вторым

Пример с ценой на телефоны (указывается в регистрации)

Могут использоваться вместе или поотдельности

**5. Какие типы маршрутизируемых событий есть в WPF (поясните каждый)?**

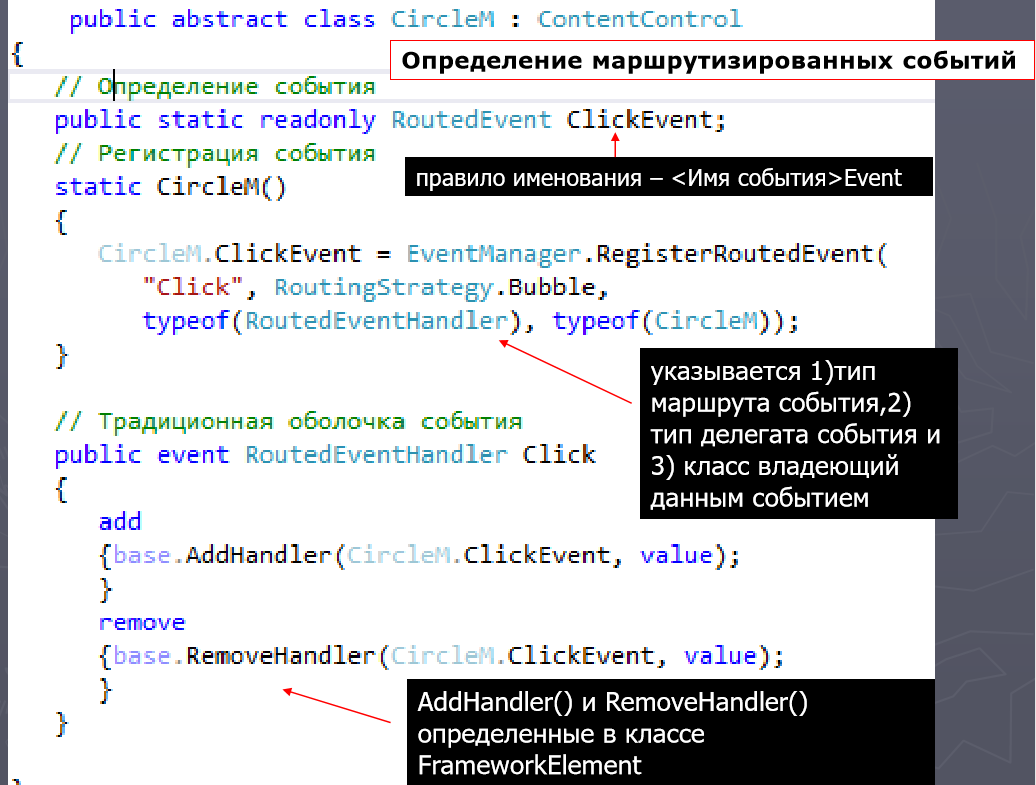
Маршрутизируемые события позволяют обработать событие в одном элементе ( например в panel ), хотя оно возникло в другом ( например в button).

Прямые (direct events) - возникают и отрабатывают 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

Опускающиеся, туннельные (tunneling events) - начинает отрабатывать в корневом элементе окна приложения и идет далее по вложенным элементам, пока не достигнет элемента, вызвавшего это событие

**6 Как создать RoutedEvent?**

****

**7. Поясните концепцию Command в WPF? В чем ее преимущества?**

-Назначение: - использовать одну и ту же команду для нескольких ЭУ

-абстрагировать набор действий от конкретных событий конкретных элементов

Суть паттерна комманда Суть: Представить запрос как ООП объект (не метод)  Конфигурация команды  Для определения и выполнения в разное время: создания очереди, stop и ….  undo, redo  Протоколирования и структурирования системы

**8. Как используются команды**

Все команды реализуют интерфейс System.Window.Input.ICommand:

public interface ICommand

{

    event EventHandler CanExecuteChanged;

    void Execute (object parameter);

    bool CanExecute (object parameter);

}

 public class RelayCommand : ICommand

    {

        private Action<object> execute;

        private Func<object, bool> canExecute;

        public event EventHandler CanExecuteChanged

        {

            add { CommandManager.RequerySuggested += value; }

            remove { CommandManager.RequerySuggested -= value; }

        }

        public RelayCommand(Action<object> execute, Func<object, bool> canExecute = null)

        {

            this.execute = execute;

            this.canExecute = canExecute;

        }

        public bool CanExecute(object parameter)

        {

            return this.canExecute == null || this.canExecute(parameter);

        }

        public void Execute(object parameter)

        {

            this.execute(parameter);

        }

    }

**9. Как создать RoutedUICommand?**

Класс RoutedUICommand имеет несколько конструкторов. Экземпляр RoutedUICommand можно создавать без дополнительной информации, однако практически всегда необходимо задавать имя команды, текст команды и тип владения. Вдобавок может предоставляться сокращенная клавиатурная комбинация для помещения в коллекцию InputGestures

// Создание команды requery

private static RoutedUICommand requery;

static MyCommands()

{

// Инициализация команды

InputGestureCollection inputs = new InputGestureCollection();

inputs.Add(new KeyGesture(Key.R, ModifierKeys.Control, "Ctrl + R"));

requery = new RoutedUICommand("Requery", "Requery", typeof(MyCommands), inputs);

}

public static RoutedUICommand Requery

{

get { return requery; }

}