1. Свойство зависимости – набор служб, которые можно использовать для расширения функциональных возможностей свойства. Так же довольно часто используемые при работе с анимацией и привязкой данных.
2. Если мы хотим применять свойства зависимостей, то нам надо унаследовать свой класс от абстрактного класса DependencyObject. В нашем классе мы определяем два свойства зависимостей: TitleProperty и PriceProperty. Обратите внимание, что они объявляются с модификаторами public static readonly.

public static readonly DependencyProperty TitleProperty;

public static readonly DependencyProperty PriceProperty;

Затем свойства регистрируются в статическом конструкторе.

3,4) WPF предоставляет два способа валидации значения свойства:

ValidateValueCallback: делегат, который возвращает true, если значение проходит валидацию, и false - если не проходит

CoerceValueCallback: делегат, который может подкорректировать уже существующее значение свойства, если оно вдруг не попадает в диапазон допустимых значений

При установке значения сначала срабатывает делегат ValidateValueCallback, который возвращает true, если значение проходит валидацию. Далее срабатывает делегат CoerceValueCallback, который может модифицировать это значение, если оно вдруг является некорректным.

5)Прямое, туннельное, пузырьковое.

Прямое событие аналогично событиям в формах Windows, которые создаются элементом, в котором происходит событие. Можно использовать в триггерах.

Хорошим примером прямого события будет событие MouseEnter.

Туннельное - Вызываются обработчики событий в корне дерева элементов, и затем событие перемещается по визуальному дереву ко всем дочерним узлам, пока не достигнет элемента, в котором возникло событие.

Пузырьковое - Событие барботирования начинается с элемента, из которого происходит событие. Затем он перемещается вверх по визуальному дереву к верхнему элементу визуального дерева. Итак, в WPF самый верхний элемент – это, скорее всего, окно.

6) Для определения маршрутизированных событий в классе создвалось статическое поле по типу RoutedEvent:

public static RoutedEvent СобытиеEvent

Это поле, как правило, имеет суффикс Event. Затем это событие регистрируется в статическом конструкторе.

И также класс, в котором создается событие, как правило определяет объект-обертку над обычным событием. В этой обертке с помощью метода AddHandler происходит добавление обработчика для данного события, а с помощью метода RemoveHandler - удаление обработчика.

7) Команды представляют механизм выполнения какой-нибудь задачи.

9)

В WPF определено много команд, но даже их может оказаться недостаточно. Поэтому нередко разработчики создают свои собственные команды. Наиболее простой способ создания команды - использование готовых классов RoutedCommand и RoutedUICommand, в которых уже реализован интерфейс ICommand. Итак, в файле кода под классом окна создадим новый класс, назовем его WindowCommands, который будет содержать новые команды:

using System.Windows.Input;

public class WindowCommands

{

static WindowCommands()

{

Exit = new RoutedCommand("Exit", typeof(MainWindow));

}

public static RoutedCommand Exit { get; set; }

}

В данном случае команда называется Exit и представляет собой объект RoutedCommand. В конструктор этого объекта в данном случае передается название команды и элемент-владелец команды (здесь объект MainWindow). Причем команда определяется как статическое свойство.

Теперь в коде XAML установим привязку к этой команде и задействуем ее.