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

DependencyProperty – механизм WPF, позволяющий означить свойства визуальных контролей через XAML. Многие свойства, используемые в XAML разметке – на самом деле DependencyProperty.

Использование

DependencyProperty нужны для:

* Data Binding. Биндинг возможен только для DependencyProperty.
* Анимаций.
* Стилей и триггеров.
* Они появляются в окне свойств контрола в Visual Studio

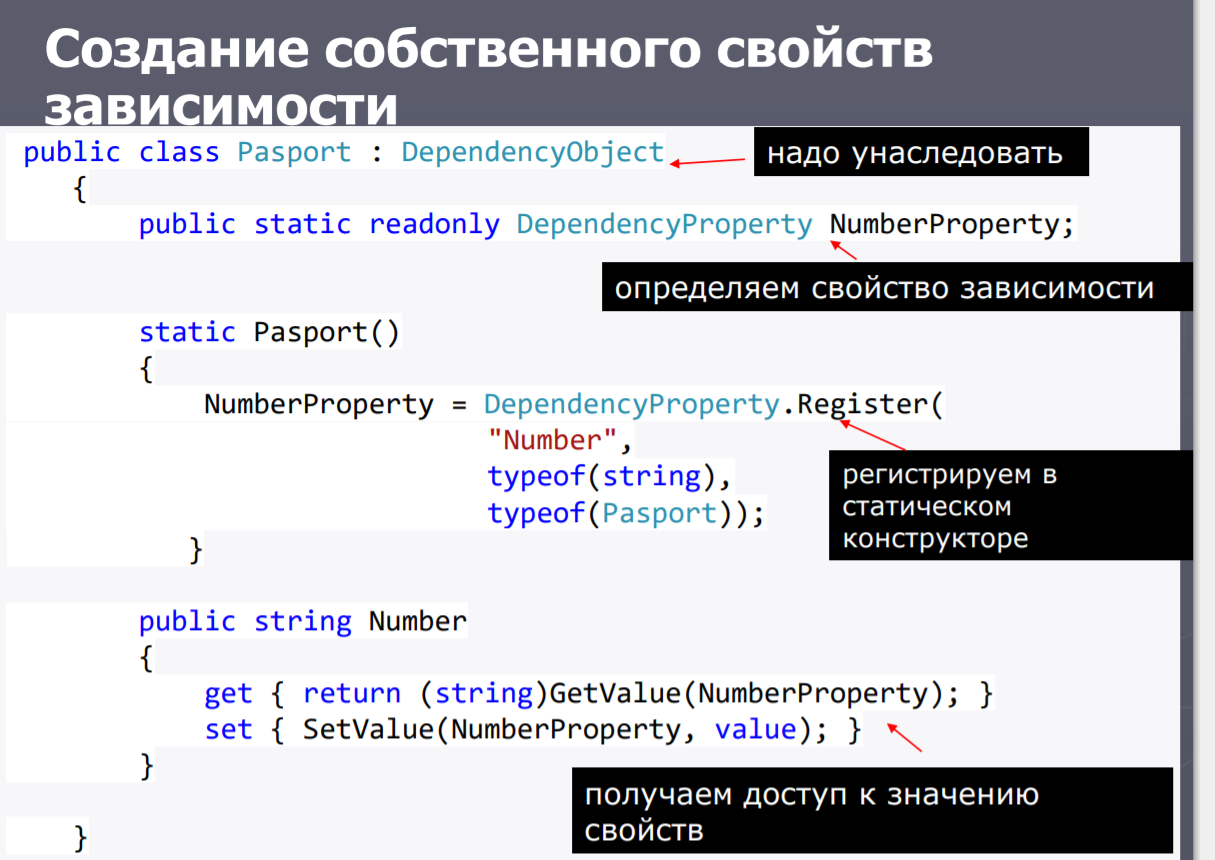
Windows Presentation Foundation (WPF) предоставляет набор служб, которые можно использовать для расширения функциональных возможностей свойства. В совокупности эти службы обычно называются системой свойств WPF. Свойство, поддерживаемое системой свойств WPF, называется свойством зависимостей.

DependencyProperty (Свойство зависимости) — это основа любого Silverlight/WPF/WP7 приложения. Их можно рассматривать как обычные свойства, но они обладают дополнительными возможностями, позволяющими работать с анимацией, привязкой данных, стилями. Также они эффективно потребляют память и поддерживают уведомления об изменениях.

Одной из основной черт DependecyProperty является приоретизация. Т.е. при присвоении им значения будет учтено несколько факторов. Наивысший приоритет будет иметь анимация, самый низкий – значения по умолчанию. Более подробно можно посмотреть на MSDN.

В принципе, запоминать последовательность приоритетов не очень нужно — она и так интуитивно понятна. Для того чтобы в классе можно было реализовывать Dependency Property необходимо, чтобы класс наследовался от DependencyObject либо его наследников, таких, например, как UIElement, Geometry, FrameworkTemplate, Style и других.

1. **Как создать DependencyProperty?**



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

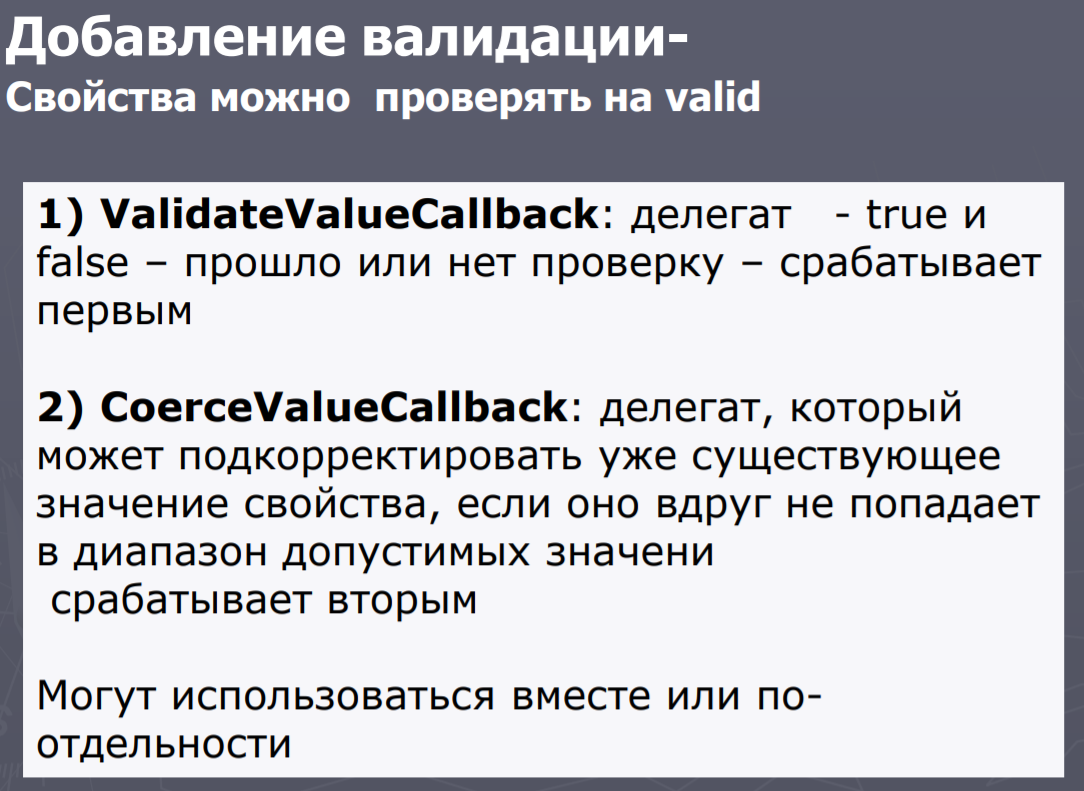
Этот обратный вызов может принимать или отбрасывать новые значения. Обычно он применяется для обнаружения очевидных ошибок, которые нарушают ограничения свойства. Его можно передать в качестве аргумента при вызове метода DependencyProperty.Register().

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

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

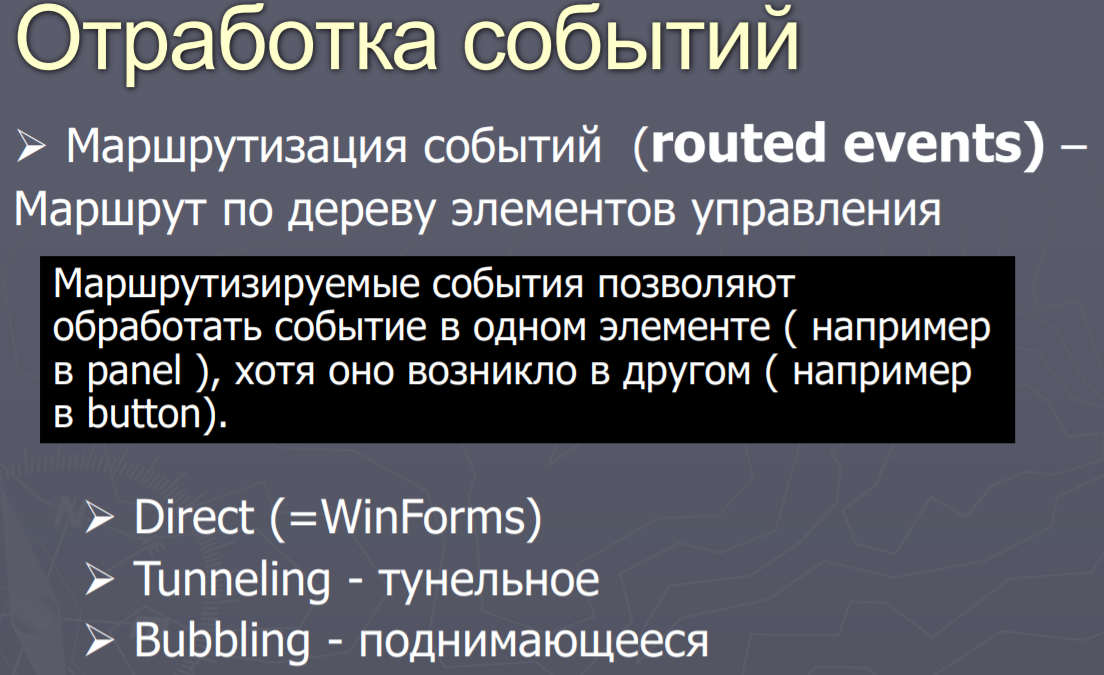
Этот обратный вызов может изменять введенные значения на более приемлемые. Обычно он применяется для обработки конфликтов между значениями свойств зависимости, установленных для одного и того же объекта. Такие значения могут быть верны порознь, но противоречить друг другу. Для использования этого обратного вызова передайте его в качестве аргумента конструктора при создании объекта FrameworkPropertyMetadata, который затем передается методу DependencyProperty.Register().

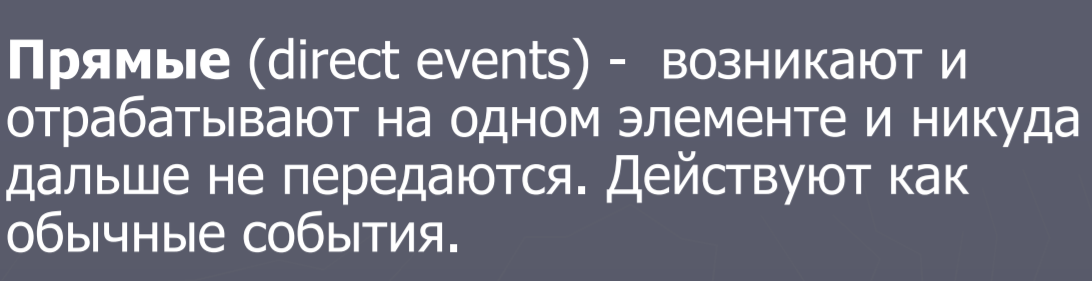
CoerceValueCallback: делегат, который может подкорректировать уже существующее значение свойства, если оно вдруг не попадает в диапазон допустимых значений

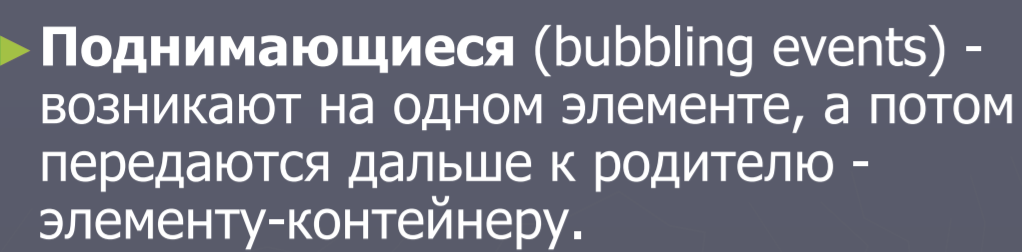


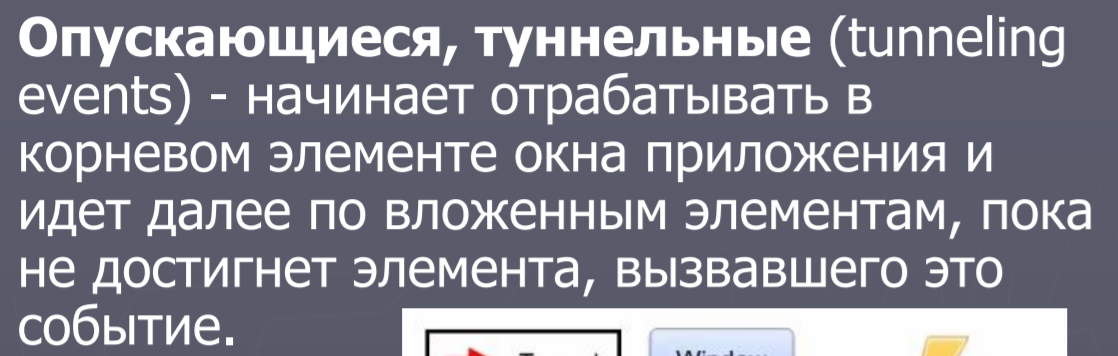
**5. Какие типы маршрутизируемых событий есть в WPF (поясните**

**каждый)?**

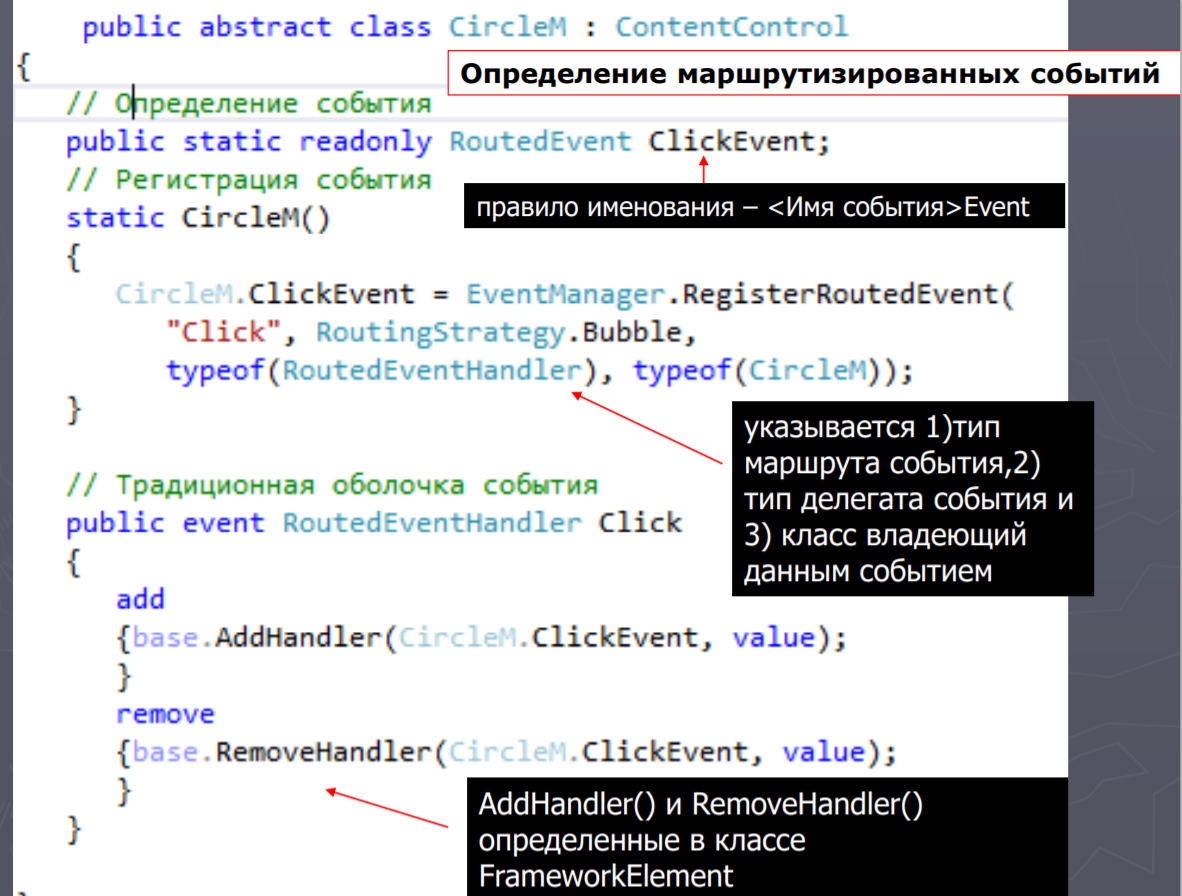




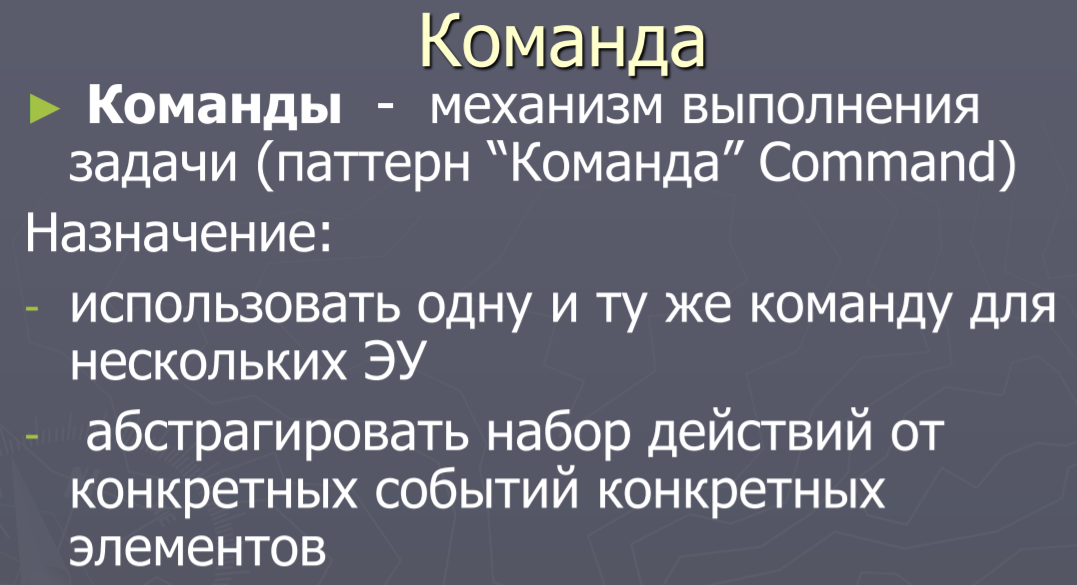


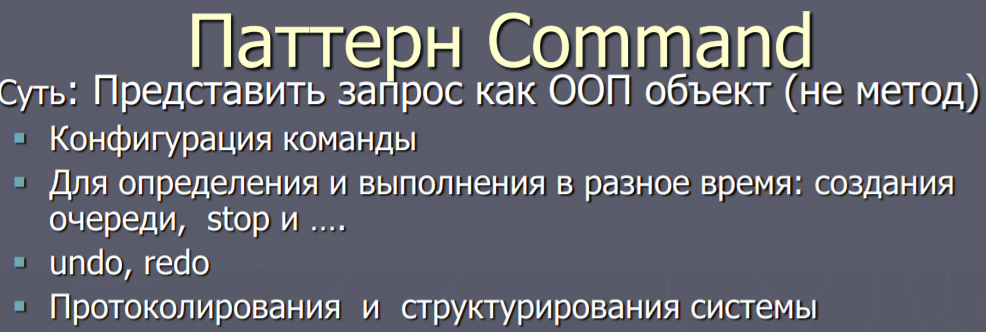


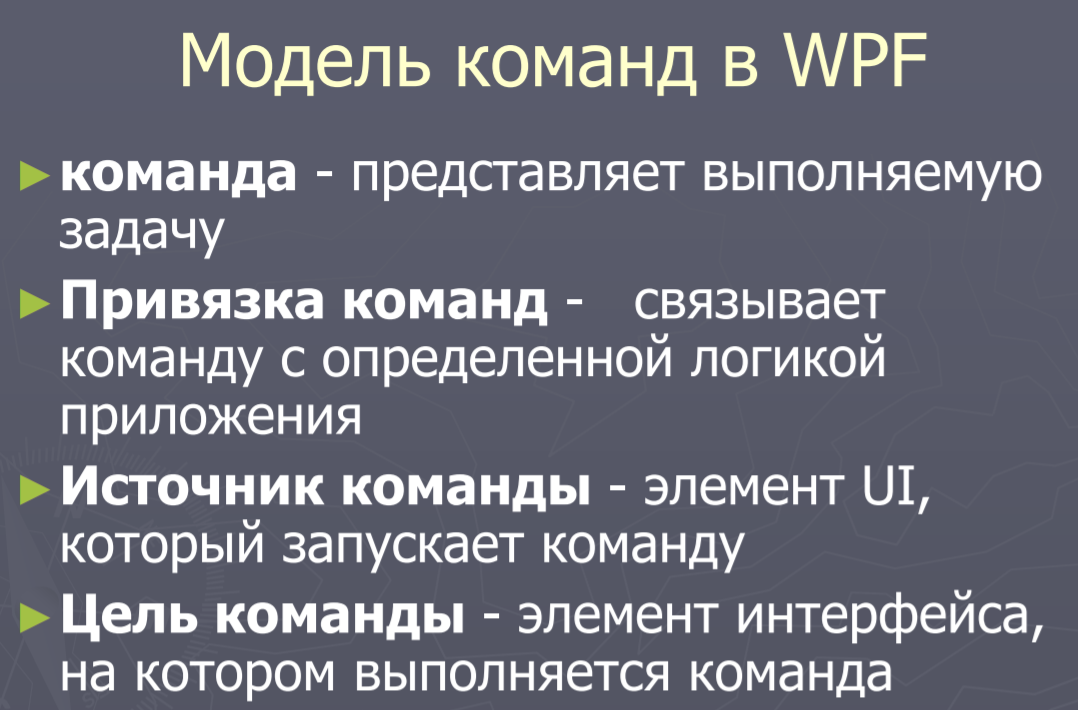
**6. Как создать RoutedEvent?**

****

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







8. Как используются команды?

**9. Как создать RoutedUICommand?**

Большинство команд, с которыми придется иметь дело, будут представлять собой не объекты RoutedCommand, а экземпляры класса **RoutedUICommand**, унаследованного от RoutedCommand. (В действительности все заготовленные команды, которые предлагаются в WPF, являются объектами RoutedUICommand.)

Класс RoutedUICommand предназначен для команд с текстом, который должен отображаться где-нибудь в пользовательском интерфейсе (например, текстом для элемента меню или текстом подсказки для кнопки в панели инструментов). Класс RoutedUICommand добавляет одно свойство Text, в котором указывается текст, отображаемый вместе с данной командой.

Преимущество подхода с определением текста вместе с командой (а не непосредственно в элементе управления) состоит в том, что это позволяет производить локализацию в одном месте. Если же текст команды никогда не отображается где-либо в пользовательском интерфейсе, то вполне подойдет и класс RoutedCommand.

Отображать в пользовательском интерфейсе текст RoutedUICommand вовсе не обязательно. Часто могут быть веские причины для использования какого-то другого текста. Например, вместо текста "Print Document" (Печать документа) может применяться просто "Print" (Печать), а в некоторых случаях текст вообще заменяется небольшим графическим изображением.