1. Что такое MVC, MVP, MVVM?

Наиболее распространенные виды MVC-паттерна, это:

* **Model-View-Controller**

Основная идея этого паттерна в том, что и контроллер и представление зависят от модели, но модель никак не зависит от этих двух компонент.  
  
**Признаки контроллера**

* Контроллер определяет, какие представление должно быть отображено в данный момент;
* События представления могут повлиять только на контроллер.контроллер может повлиять на модель и определить другое представление.
* Возможно несколько представлений только для одного контроллера;
* **Model-View-Presenter**

Данный подход позволяет создавать абстракцию представления. Для этого необходимо выделить интерфейс представления с определенным набором свойств и методов. Презентер, в свою очередь, получает ссылку на реализацию интерфейса, подписывается на события представления и по запросу изменяет модель.

Признаки презентера:

Двухсторонняя коммуникация с представлением;

Представление взаимодействует напрямую с презентером, путем вызова соответствующих функций или событий экземпляра презентера;

Презентер взаимодействует с View путем использования специального интерфейса, реализованного представлением;

Один экземпляр презентера связан с одним отображением.

* **Model-View-View Model**

Данный подход позволяет связывать элементы представления со свойствами и событиями View-модели. Можно утверждать, что каждый слой этого паттерна не знает о существовании другого слоя.

Признаки View-модели:

Двухсторонняя коммуникация с представлением;

View-модель — это абстракция представления. Обычно означает, что свойства представления совпадают со свойствами View-модели / модели

View-модель не имеет ссылки на интерфейс представления (IView). Изменение состояния View-модели автоматически изменяет представление и наоборот, поскольку используется механизм связывания данных (Bindings)

Один экземпляр View-модели связан с одним отображением.

1. Объясните из каких компонентов состоит паттерн MVVM?

MVVM — это паттерн разработки, позволяющий разделить приложение на три функциональные части:

Model — основная логика программы (работа с данными, вычисления, запросы и так далее)

View — вид или представление (пользовательский интерфейс)

ViewModel — модель представления, которая служит прослойкой между View и Model

3. Каким образом паттерн можно реализовать в WPF?

4. Объясните назначение интерфейса INotifyPropertyChanged?

**Интерфейс** **INotifyPropertyChanged** используется для уведомления клиентов, как правило, привязки клиентов о том, что значение свойства изменилось. Например, рассмотрим Person объект со свойством с именем FirstName. Чтобы предоставить универсальное уведомление об изменении свойства, Person тип реализует **INotifyPropertyChanged** **интерфейс** и вызывает PropertyChanged событие при FirstName изменении.

5. Зачем нужен интерфейс ICommand и как паттерн MVVM использует команды в WPF-приложениях.

6. Чем отличаются команды от событий?

*События*

* Обработчик события получает ссылку на контрол в sender и все необходимые данные о событии в EventArgs. Далее можно делать с ними что требуется: менять поведение контрола, или слать данные куда подальше. Набор входных данных фиксирован и не кастомизируется.
* Событие вызывается тогда, когда нам надо, нажата клавиша, наведена мышь, и так далее. Есть много разных событий, которые могут подойти практически под любой вид задач.

*Но*

* Находятся все в одном классе
* Сложно тестировать

*Команды*

* Находятся где угодно, куда можно ссылаться при объявлении Binding, по умолчанию там, куда ссылается DataContext.
* В команду можно передать любой аргумент.
* Команду можно вызвать когда надо через прямое подключение, например к кнопке, или через KeyBindings.
* Разрешено ли выполнять команду, узнать можно до начала ее выполнения, условие помещается в CanExecute. При этом многие контролы имеют встроенную реакцию на CanExecute, и например сами могут переводить свое состояние в Disabled, если CanExecute вернет false (например MenuItem или Button). Это удобно.
* Легко тестировать просто вызвав команду отдельно из тестового метода.

7. Расскажите о назначении и об основных возможностях следующих библиотек и фреймворков: Light Toolkit, Catel и Prism

MVVM Light - это реализация MVVM. Это не единственная реализация, но теперь она признана лучшей.  
MVVM Light разработан так, чтобы быть очень гибким, и в нем уделяется большое внимание реализации. Он реализует базовую структуру шаблона проектирования MVVM, например класс ViewModelBase, и реализует универсальную Command <T>, которая является основой MVVM.

Основная цель этого набора инструментов - ускорить разработку и разработку приложений MVVM в Windows Universal, WPF, Silverlight, Xamarin.iOS, Xamarin.Android и Xamarin.Forms.

MVVM Light Toolkit может помочь нам отделить представление от модели и создать простые и легкие в обслуживании и расширении приложения. Он также создает тестируемые приложения и позволяет иметь несколько тонких слоев пользовательского интерфейса (которые сложнее тестировать автоматически).  
В инструментарии подчеркивается "возможность проектирования" (то есть возможность открывать и редактировать пользовательский интерфейс в Blend) в приложениях, которые он создает, включая создание данных времени разработки, позволяющих пользователям Blend Умеет выполнять «визуализацию» работы с элементами управления данными.

Как и другие WPF фреймворки (Prism, MVVM Light, Caliburn.Micro и т.д.), Catel предоставляет удобные средства для построения приложений в MVVM стиле.  
Главные компоненты:

* IoC (интегрированный с MVVM компонентами)
* ModelBase: базовый класс, предоставляющий автоматическую реализацию PropertyChanged (особенно в связке с Catel.Fody), сериализацию и BeginEdit/CancelEdit/EndEdit (классические "применить"/"отмена").
* ViewModelBase, умеющий привязываться к модели, оборачивая ее свойства.
* Работа с представлениями (views), которые умеют автоматически создавать и привязываться к ViewModel. Поддерживаются [вложенные контролы](https://docs.catelproject.com/vnext/introduction/mvvm/introduction-to-nested-user-controls-problem/).

Если вы разрабатываете приложения для XAML-based платформ, таких как WPF, Silverlight, Windows Phone, или Windows Store, то Prism, определённо, сильно поможет вам упростить их разработку. После создания нескольких проектов с использованием Prism, я убедился в работоспособности такого подхода и решил подготовить перевод официального руководства Prism.

Prism является руководством, разработанным для того, чтобы помочь проектировать и создавать насыщенные, гибкие и легко поддерживаемые Windows Presentation Foundation (WPF) приложения, Silverlight Rich Internet Applications (RIAs), и программы под Windows Phone 7-8, а также Windows Store приложений. Используя паттерны проектирования, которые воплощают важные принципы архитектурного дизайна, такие как [разделение ответственности (Separation of Concerns, SoC)](http://en.wikipedia.org/wiki/Separation_of_concerns) и [слабая связанность (Loose coupling)](http://en.wikipedia.org/wiki/Loose_coupling), Prism помогает вам проектировать и писать приложения со слабо связанными компонентами, которые могут независимо развиваться и потом объединяться в одно целое с минимальными усилиями. Этот тип приложений известен как составные приложения.