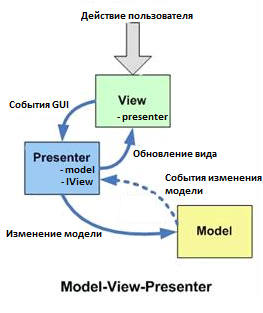
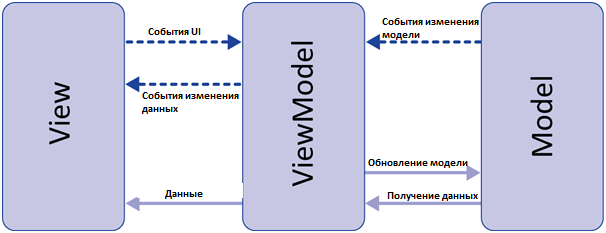
1. **Что такое MVC, MVP, MVVM?**

Model-View-Presenter

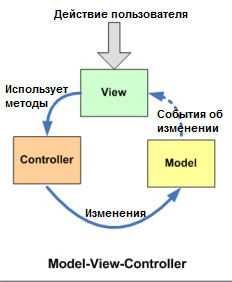


Данный подход позволяет создавать абстракцию представления. Для этого необходимо выделить интерфейс представления с определенным набором свойств и методов. Презентер, в свою очередь, получает ссылку на реализацию интерфейса, подписывается на события представления и по запросу изменяет модель.

###### 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.

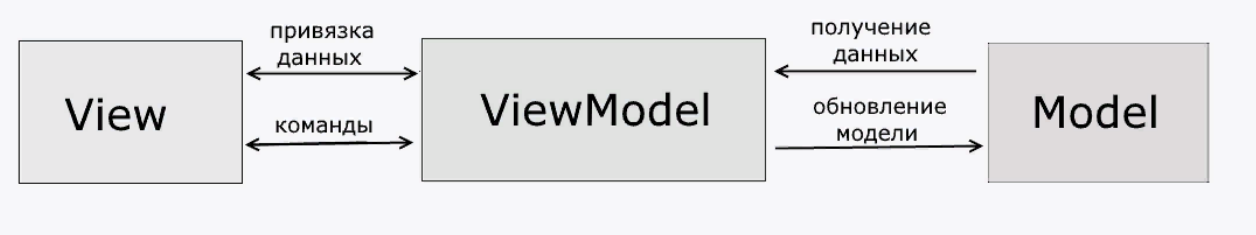
###### Model-View-Controller

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

.

1. **Объясните из каких компонентов состоит паттерн MVVM?**

MVVM состоит из трех компонентов: модели (Model), модели представления (ViewModel) и представления (View).



1. Каким образом паттерн можно реализовать в WPF?
2. **Объясните назначение интерфейса INotifyPropertyChanged?**

 Если в модели изменяются значения свойств, при реализации моделью интерфейса INotifyPropertyChanged автоматически идет изменение отображаемых данных в представлении, хотя напрямую модель и представление не связаны.

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

В MVVM не пишутся обработчики событий. Функции, которые нужно выполнять контролам, пишутся во ViewModel и биндятся к контролам точно так же, как поля. Только используется механизм команд.

Команда должна представлять из себя экземпляр класса, реализующего интерфейс ICommand.

Для привязки команды кнопки необходимо привязать свойство, которое является реализацией ICommand. В **состав ICommand**входят:

1. EventHandler CanExecuteChanged;
2. **bool**CanExecute( параметр объекта);
3. **void**Execute( параметр объекта);

**CanExecuteChanged**вызывается, когда происходят изменения, которые могут изменить ли команда может быть выполнена.

**CanExecute**определит, может ли команда быть выполнена или нет. Если он возвращает false, кнопка будет отключена на интерфейсе.

**Execute**запускает логику команды.

1. **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.

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