Una solución puede tener varios proyectos adentro, estos proyectos pueden estar programados en diferentes lenguajes de programación.

Namespaces: Funcionan para agrupar clases, métodos, atributos, elimina el problema de tener clases con mismos nombres.

Un assembly es el código compilado de una clase, es diferente a un namespaces, este ultimo puede estar compilado en varios assemblies.

Tipos de valor

* Tipos primitivos
* Enums
* Struts
* Nullables -> Asignados a los tipos primitivos

Se puede utilizar “var” para declarar una variable. Solo vale para declarar variables, no pueden modificarse tras la declaración y solo podemos usarla junto con una expresión que inicializa la variable.

Se puede utilizar el “is” como operador de relación, es como el instanceof de Java.

Se puede utilizar el “as” como un casteador.

DateTime

Format utiliza {0} para el posicionamiento de valores.

Replace, utiliza @nombre para sustituir por valores.

Declarar un enum

enum Nombre {

opción = 1,

opción = 2,

opción = 3

}

Nullable

* Nullable<tipoDeValor> variable
  + Nullable<int> numero = null; .hasValue();

.value();

* TipoDeValor? Variable
  + DateTime? fechaNac

Lambda

* Array.Where(variable -> condición).ToList();

OpenFileDialog

* FileInfo: Información sobre el archive que selecciono en el dialogo
* TextReader: Es el lector del archivo.

Opciones -> Avanzado, checkear por sobresaliente de operaciones aritméticas

checked (9876554 \* 9876554);

Clases estáticas: Pueden tener un constructor por defecto y debe ser static.

Tipos Nullable: Se le agrega el signo de pregunta. Permite asignar valores null a primitivos

Tipos ref: Cuando se utiliza ref, en vez de crear una copia en el argumento, pasas una referencia.

Tipos out: El método es quien inicializa el parámetro. En caso de que el mismo no sea inicializado.

enum Estacion {Primavera, Verano, Otoño, Invierno}

enum Estacion: **short** (A cada enum le asigna un short en vez de un int, para salvar memoria)

Se puede utilizar los operadores unarios en los enums al ser ints

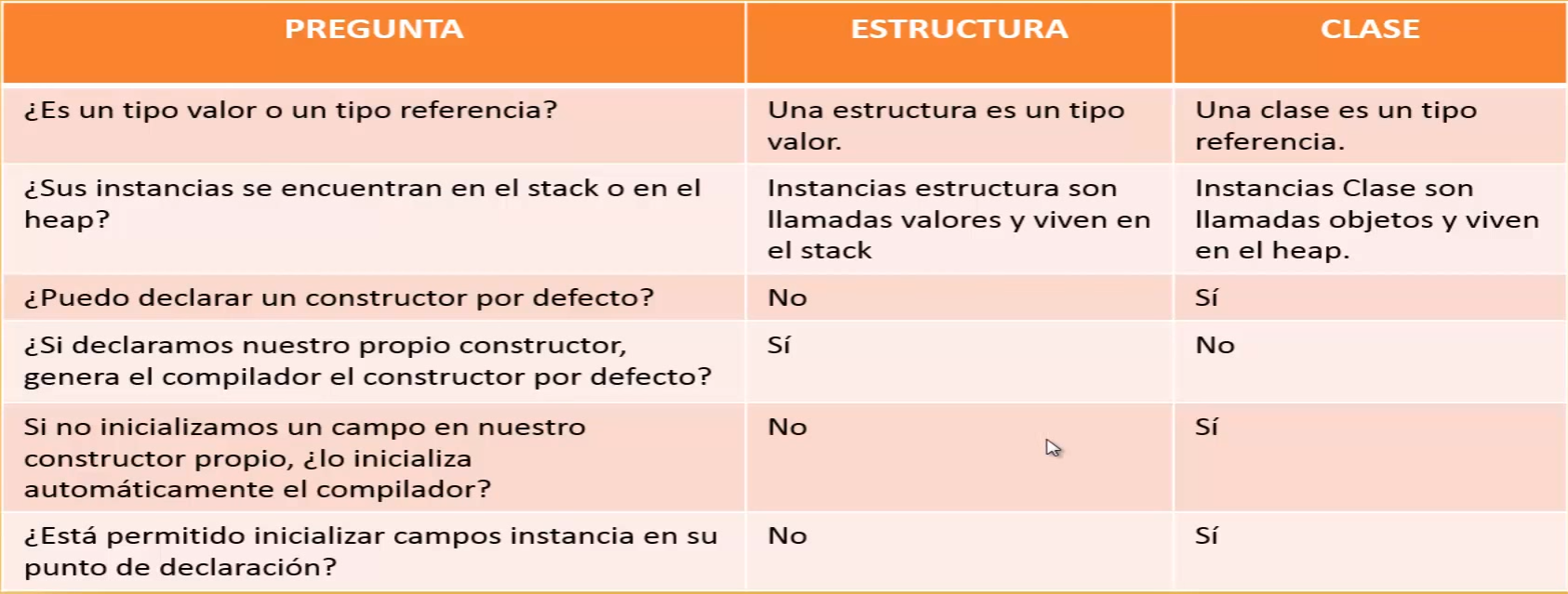
Struct utilizada para crear nuestras propias estucturas, como int, float, doublé, etc

struct Tiempo{

public int horas, minutos, segundos;

}

En caso de las estructuras, se deben inicializar sus variables y no se pueden declarar un constructor por defecto. Se pueden declarar constructores, pero se deben asegurar de inicializar todas las varaibles (campos) de la estructura. Estas variables no pueden inicializarse cuando se las declara.



Cuando se realiza una copia de una estructura, no apunta a la misma dirección al igual que en las clases. Estas se manejan de forma diferente. Ya que no se encuentran en el heap, sino en el stack

Auto auto = new Auto();

Auto auto2 = auto;

Siendo clases si se modifica uno, se modifica el otro.

Auto auto = new Auto();

Auto auto2 = auto;

Siendo estructuras, si se modifica uno, no se ve impactado el otro, se manejan de forma independiente al estar en el stack.

Data Binding: Asociar datos a controles, para cuando se cambian vistas

Patrón Model - View – ViewModel

Para el trabajo con datos

Para ello se debe trabajar con 4 temas:

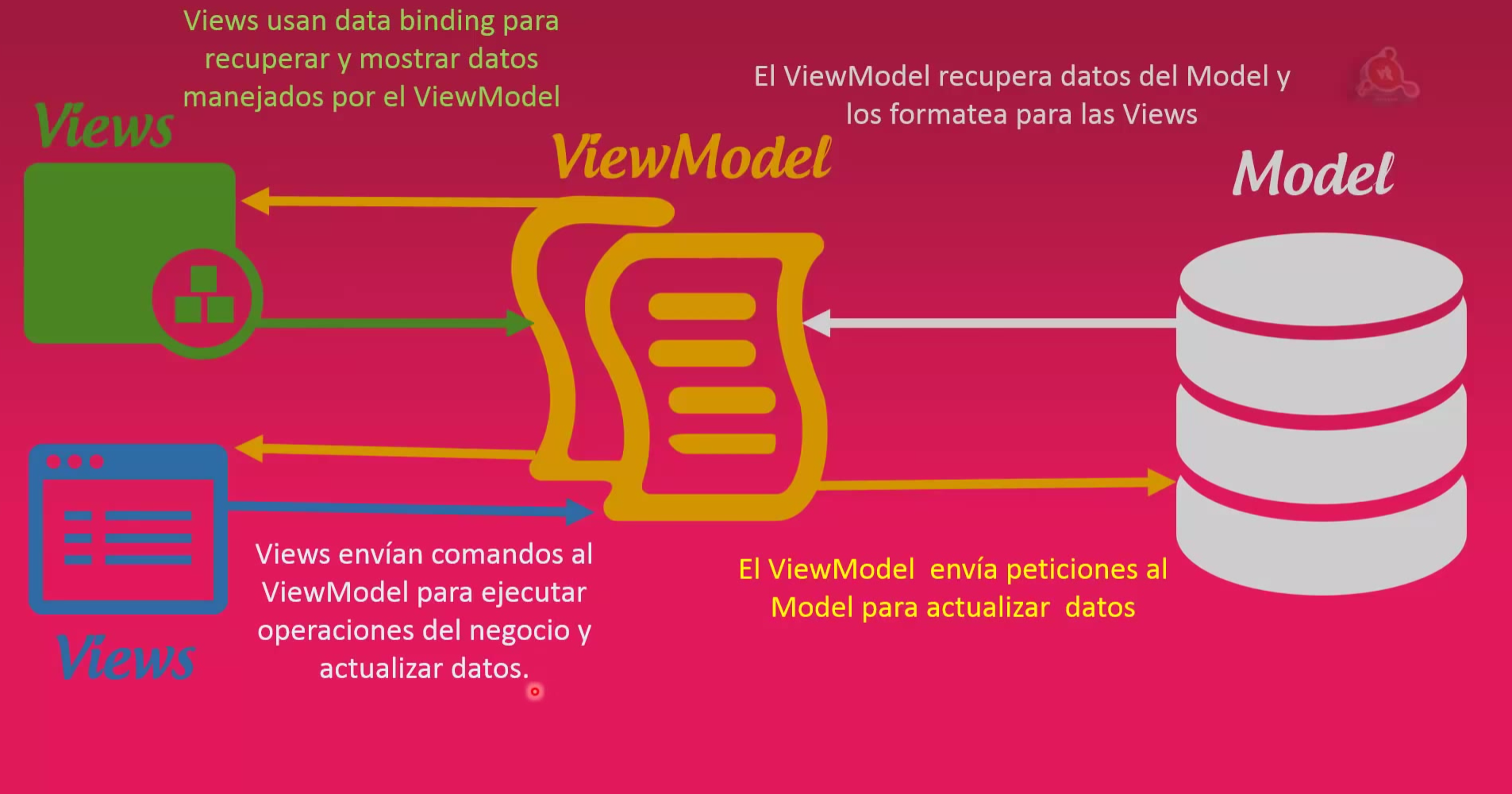
1. Data binding: Conectar Interfaz de usuario a los datos que se muestran
2. Model-View-ViewModel: Implementar modelo
3. ViewModel: Permite separar la lógica de la interfaz de usuario, del modelo de datos y de la lógica del negocio de una aplicación.
4. Search Contract: Característica de búsqueda de Windows que permite de modo sencillo incluir sistemas de búsqueda en las aplicaciones.

Patrón Model – View – ViewModel: Es un patrón de diseño en el cual:

* El model: provee los datos usados por la aplicación.
* View: Representa el modo en que los datos son mostrados en la interfaz de usuario.
* ViewModel: Es el que contiene la lógica que conecta al model y las view.

En este sistema lo que hace el ViewModel es recuperar datos del model y los formatea para que se muestren en las View y realiza peticiones al model para ir actualizando los datos.

A diferencia del MVC, es que el viewmodel permite una conexión bidireccional, es decir, las vistas también pueden enviar peticiones al view model para ejecutar operaciones del negocio y actualizar datos.



Una aplicación puede proveer diferentes vistas para los mismos datos.

El ViewModel es el encargado de formatear y asegurarse que los datos del modelo pueden ser representados en las vistas.

Data Binding: Permite enlazar una propiedad de un control a una propiedad de un objeto. En este caso, los data binding vienen a ser los ViewModel.

Utilizando el Model en el binding, indicamos si es bidireccional, pero para que se realicen los cambios hay que utilizar una interfaz INotifyPropertyChanged. Esta interfaz debe ser implementada por el modelo.

El patrón command: El ViewModel va a poder exponer métodos que la View puede invocar. Para proveer operaciones que se puedan disparar por los controles que tenga la vista.

Permite al ViewModel exponga métodos que la View puede invocar en forma de comandos, su objetivo es mantener la total independencia entre el ViewModel y el View.

Para aplicar este patrón, los ViewModel deben implementar la interfaz ICommand

Entity Framework

