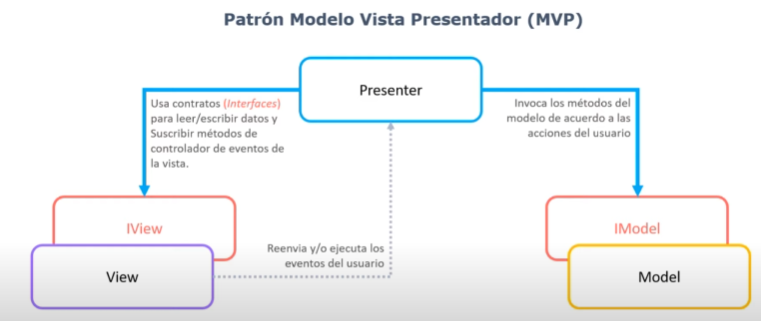


Es el único patrón que se adapta mejor a WF que MVC y MVVM.

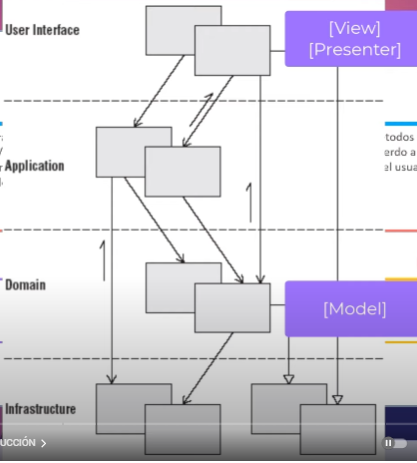
Estos 3 depende del alcance y de cómo se los esté usando pueden ser tanto patrones de arquitectura como de diseño:

Lo explica, si se los usa para toda la app sería un patrón de arquitectura, pero si se los aplica solo para la capa de presentación serían patrones de diseño.

Afirma que en softwares complejos, con capas de acceso a datos, se los deja para la capa de presentación (serían patrones de diseño).



Dice que la vista debe ser lo mas “tonta” posible.



Señala que el modelo pasaría a ser parte del dominio, y el resto de la presentación.

Validación de datos con Data Annotation Validators

Para usar esto hay que usar la referencia:



Y agregar ese using:

using System.ComponentModel.DataAnnotations;

Bah, al menos al ppio, usé esta que sigue, no la otra.

using System.ComponentModel;

DisplayName es para que muestre el nombre que se indica.

Required indica que el campo no puede estar vacío.

Deja en claro que las validaciones las hace en el modelo.

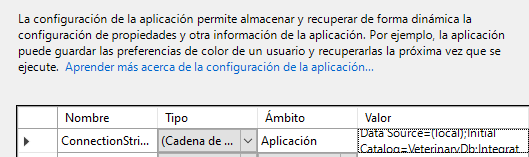
Agrega una interfaz a model. Reitera que según DIP, el principio Solid de inversión de las dependencias, los detalles deben depender de las abstracciones. La capa de acceso a datos debe depender del dominio. En dicha interfaz define los comportamientos para el acceso a datos.

En view agrega otra interfaz.

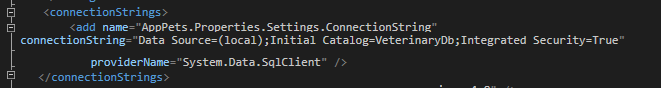
ConnectionString

Dice que se puede agregar en app.Config y quizás esa sea la mejor práctica.

Por otro lado, el lo hace desde Properties, yendo a settings.



Se agrega solito a App.Config



Para poder usar esto hay que agregar la referencia al ensamblado de configuración, así como el using en la clase program.

Inyección de dependencias

La inyección de dependencia trata de como un objeto conoce a otro objeto, dependiente, a través de la interfaz.

Permite gestionar mejor futuros cambios de código.

Singletón

Utiliza esto para que solo se abra un formulario de vista de mascota y no infinitos de ellos.

private static PetForm instance;

public static PetForm GetInstance()

{

if (instance==null || instance.IsDisposed)

{

instance = new PetForm();

}

else

{

if (instance.WindowState == FormWindowState.Minimized)

{

instance.WindowState = FormWindowState.Normal;

}

instance.BringToFront();

}

return instance;

}

Esto lo hace en la clase que se va a instanciar (una sola vez). Luego esto, en la clase que crea la instancia de esa clase:

IPetView view = PetForm.GetInstance();

IPetRepository repository = new PetRepository(sqlConnectionString);

new PetPresenter(view, repository);

Formulario como contenedor MDI

1. Marcar en el formulario padre IsMdiContainer true;
2. Usa el método GetInstance del formulario secundario, se le pasa por parámetro el formulario contenedor padre.

public static PetForm GetInstance(Form parentContainer)

{

if (instance==null || instance.IsDisposed)

{

instance = new PetForm();

instance.MdiParent = parentContainer;

}

1. Donde se instancia esa clase se pone:

IPetView view = PetForm.GetInstance((MainView)mainView);

1. Para que ocupe todo el espacio y no tenga bordes:

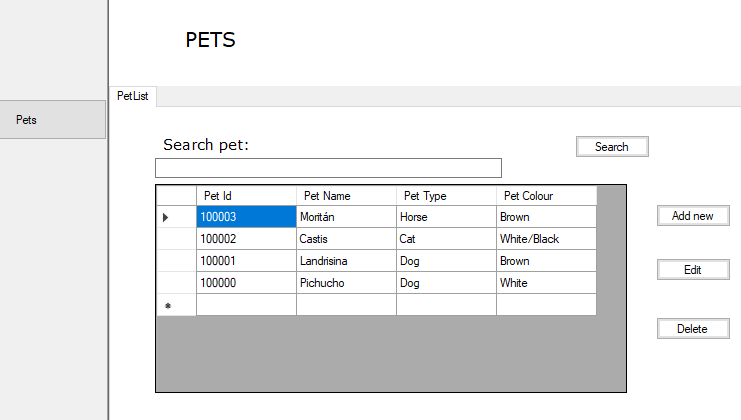
if (instance==null || instance.IsDisposed){

instance = new PetForm();

instance.MdiParent = parentContainer;

instance.FormBorderStyle = FormBorderStyle.None;

instance.Dock = DockStyle.Fill; }



Responsividad

Selecciona los botones y les modifica el anchor. Estaban en Top,Left y les pone Top, Right.

Al textbox le pne top, right y left.

Al dataGridView le pone los 4 costados.

Delegados

Con los delegados asocia los eventos con los botones.

+Validaciones

Agrega una clase common al presentador