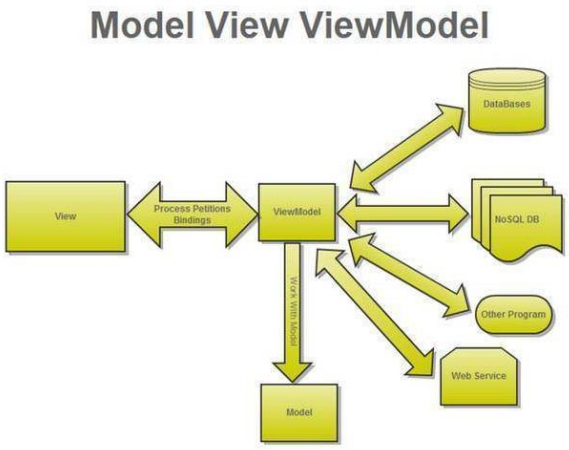
EJEMPLO PATRÓN MVMN

En el patrón de diseño Modelo-Vista-Controlador (MVC) o en su versión adaptada para Kotlin, Modelo-Vista-ViewModel (MVVM), la idea es organizar tu código en tres componentes principales:

- **Modelo**: Representa los datos.
- **Vista**: Se encarga de la presentación y la interfaz de usuario (va a pintar los datos en la pantalla)
- **ViewModel** (o Controlador en MVC): Actúa como intermediario entre el Modelo y la Vista, se encarga de recuperar/persistir los datos. Cabe resaltar que los cambios en el *ViewModel* influyen directamente en la vista y viceversa.



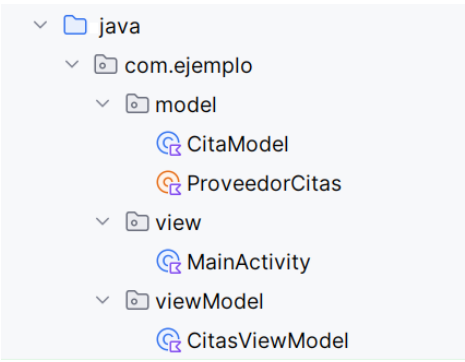
Estos nos permiten, de cierta forma, escribir el código de nuestra app a partir de tres capas o partes lógicas que se relacionan entre sí. No obstante, el principal propósito de MVVM es buscar **mejorar y facilitar la modificación del código en caso de que se presente algún error** o debamos realizarle mantenimiento al *software*.

A continuación, se ve un ejemplo simple de cómo podrías organizar una aplicación sencilla en Kotlin siguiendo el patrón **MVVM**.

Primero, crearemos un proyecto en Android Studio del tipo "**Empty Views Activity**", sobre el que iremos creando las diferentes clases que se muestran a continuación:

Supongamos que estás desarrollando una aplicación donde se muestra una cita y su autor, que podrá ser diferente cada vez que se abre.

La estructura de paquetes podría verse así:



Modelo (com.ejemplo.model): Aquí defines las clases que representan tus datos. Por ejemplo, una clase CitaModel que representa una cita.

```
package com.ejemplo.model

data class CitaModel (val cita:String, val autor:String)
```

ViewModel (com.ejemplo.viewModel): Aquí defines el código que recupera los datos, en nuestro caso los obtiene del proveedor de citas.

```
package com.ejemplo.viewModel

import androidx.lifecycle.ViewModel
import com.ejemplo.model.CitaModel
import com.ejemplo.model.ProveedorCitas

class CitasViewModel: ViewModel() {

    //Lógica para interactuar con el Modelo y preparar datos
    var cita: CitaModel

    init {
        cita = ProveedorCitas.getCitaRandom()
    }
}
```

Clase **ProveedorCitas** (com.ejemplo.model), a través del cual se genera una cita de manera aleatoria.

```
package com.ejemplo.model

object ProveedorCitas {
    fun getCitaRandom(): CitaModel {
        val position = (0 ≤ until < citas.size).random()
        return citas[position]
    }

    private val citas = listOf(
        CitaModel(
            cita = "If debugging is the process of removing software bugs, then programming must be the process of putting them in",
            autor = "Edsger Dijkstra"
        ),
        CitaModel(
            cita = "A user interface is like a joke. If you have to explain it, it's not that good.",
            autor = "Anonymous"
        ),
        CitaModel(
            cita = "I don't care if it works on your machine! We are not shipping your machine!",
            autor = "Vidiu Platon"
        ),
        CitaModel(
            cita = "Measuring programming progress by lines of code is like measuring aircraft building progress by weight.",
            autor = "Bill Gates"
        )
    )
}
```

Vista (com.ejemplo.view): Aquí defines las actividades, fragmentos o cualquier otra interfaz de usuario. Se cargan en la pantalla los datos que se han recuperado en el ViewModel.

En la vista se recupera el viewModel usando una función llamada ***by***. La función ***by*** proporciona una manera conveniente de crear propiedades que se inicializan de forma lazy. Esto significa que la propiedad se calculará e inicializará solo cuando se acceda a ella por primera vez, los accesos posteriores a la propiedad devolverán el valor almacenado en caché.

Un viewModel no lo podemos crear cuando nosotros queramos, sino que es Android quien decide cuando crearlo, por eso usamos la función viewModels() para pedir que nos devuelva el acceso a la viewModel creada desde la factoria de viewModels.

Por lo general, solicitas un ViewModel la primera vez que el sistema llama al método onCreate() del objeto de una actividad. El sistema puede llamar a onCreate() varias veces durante la vida de una actividad, como cuando rota la pantalla de un dispositivo.

El ViewModel existe desde la primera vez que solicitas un ViewModel hasta que finaliza la actividad y se destruye.

A continuación, se muestra un ejemplo de Activity usando **findViewById** para buscar un componente determinado por su id dentro de un layout especifico. En este caso dentro del layout cargado a través del método: setContentView(R. layout. activity_main)

MainActivity.kt

```kotlin
package com.ejemplo.view

import android.os.Bundle
import android.widget.TextView
import androidx.activity.viewModels
import androidx.appcompat.app.AppCompatActivity
import com.ejemplo.viewModel.CitasViewModel
import com.maestre.ejemplomvvm.R

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        //recuperar viewModel
        val viewModel: CitasViewModel by viewModels()

        //Pintar pantallas
        val valorCita = findViewById<TextView>(R.id.tvCita)
        valorCita.text = viewModel.cita.cita

        val valorAutor = findViewById<TextView>(R.id.tvAutor)
        valorAutor.text = viewModel.cita.autor
    }
}
```

El uso de findViewById ha sido sustituido por el uso de **ViewBinding** que es la nueva de manera de trabajar con referencias a los componentes de la vista.

Lo primero que tenemos que hacer es añadir en el gradle a nivel de módulo, dentro de la etiqueta android lo siguiente, para poder utilizar esta característica:

```
buildFeatures{
    viewBinding = true
}
```

Como hemos cambiado el gradle, nos va a pedir sincronizar, una vez termine la sincronización ya podríamos usar ViewBinding:

MainActivity.kt

```kotlin
package com.maestre.mvvmbinding.view

import android.os.Bundle
import androidx.activity.viewModels
import androidx.appcompat.app.AppCompatActivity
import com.ejemplo.viewModel.CitasViewModel
import com.maestre.mvvmbinding.databinding.ActivityMainBinding

class MainActivity : AppCompatActivity() {

    private lateinit var binding: ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        //enlazar con layout usando binding
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)
        //recuperar viewModel
        val viewModel: CitasViewModel by viewModels()
        //pintar pantalla
        binding.tvCita.text=viewModel.cita.cita
        binding.tvAutor.text=viewModel.cita.autor
    }
}
```