Facultad de Ingeniería

Desarrollo de Software para Móviles

Desafío Practico #1

ELABORADO POR:

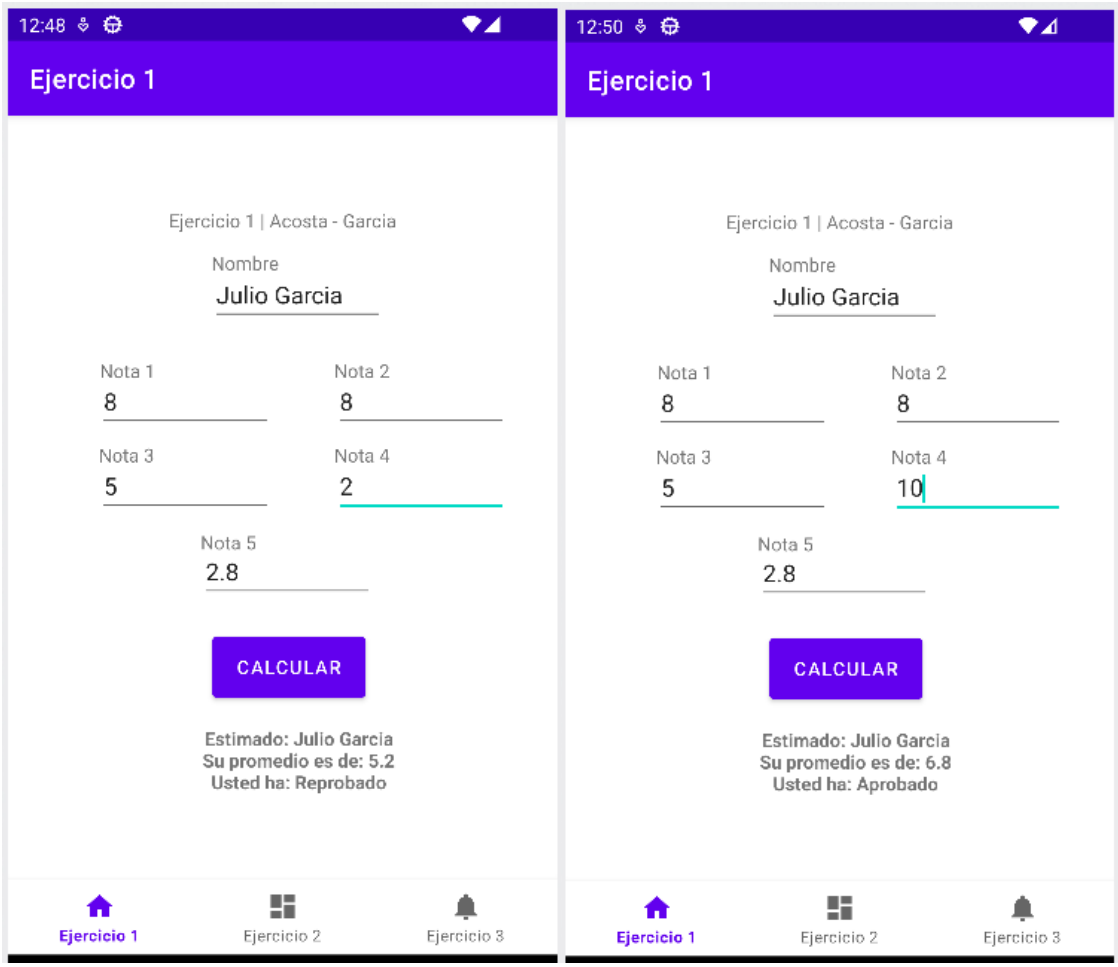
Wilfredo José Acosta Beltrán AB181924

Julio Armando García Sánchez GS181936

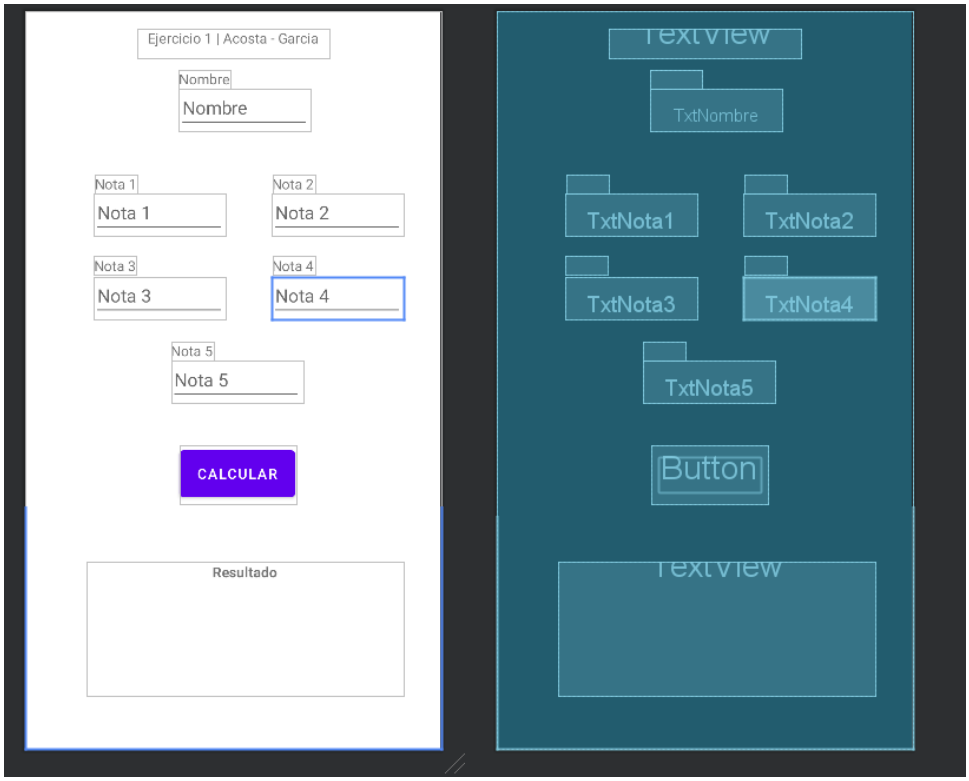
Enlace video funcionamiento

Capturas Funcionamiento y descripción de cada Ejercicio

Ejercicio 1 (100%)



Diseño de layouts

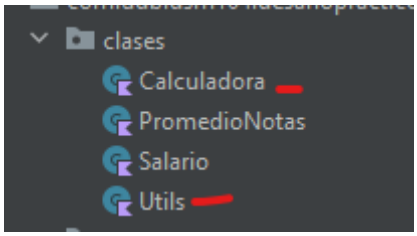


Registro de títulos a utilizar dentro de la aplicación

```xml
<resources>
    <string name="app_name">DesafioPractico1</string>
    <string name="title_home">Ejercicio 1</string>
    <string name="title_dashboard">Ejercicio 2</string>
    <string name="title_notifications">Ejercicio 3</string>

    <string name="tituloEjercicio1">Ejercicio 1 | Acosta - Garcia</string>
    <string name="tituloEjercicio2">Ejercicio 2 | Acosta - Garcia</string>
    <string name="tituloEjercicio3">Ejercicio 3 | Acosta - Garcia</string>

    <string name="msgNota1">Nota 1</string>
    <string name="msgNota2">Nota 2</string>
    <string name="msgNota3">Nota 3</string>
    <string name="msgNota4">Nota 4</string>
    <string name="msgNota5">Nota 5</string>

    <string name="nombre">Nombre</string>

    <string name="salario">Salario</string>

    <string name="calcular">Calcular</string>
</resources>
```

Para la solución de este ejercicio se creó la clase de PromedioNotas donde se almacenan y estructuras los datos solicitados en el formulario, y además de realizar las operaciones, para luego solo ser utilizado en el Fragment principal.

Igualmente se creó una clase de Utils para algunos métodos que se usan.

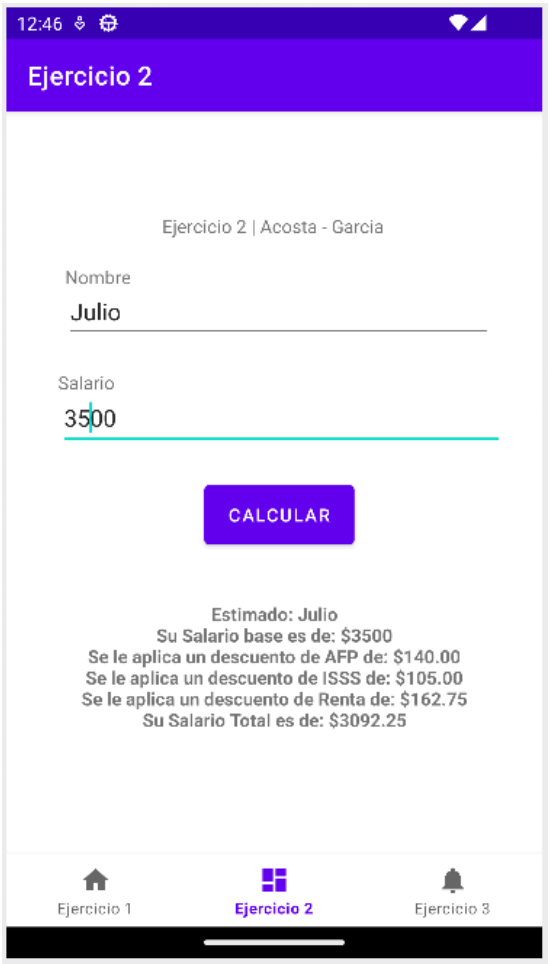


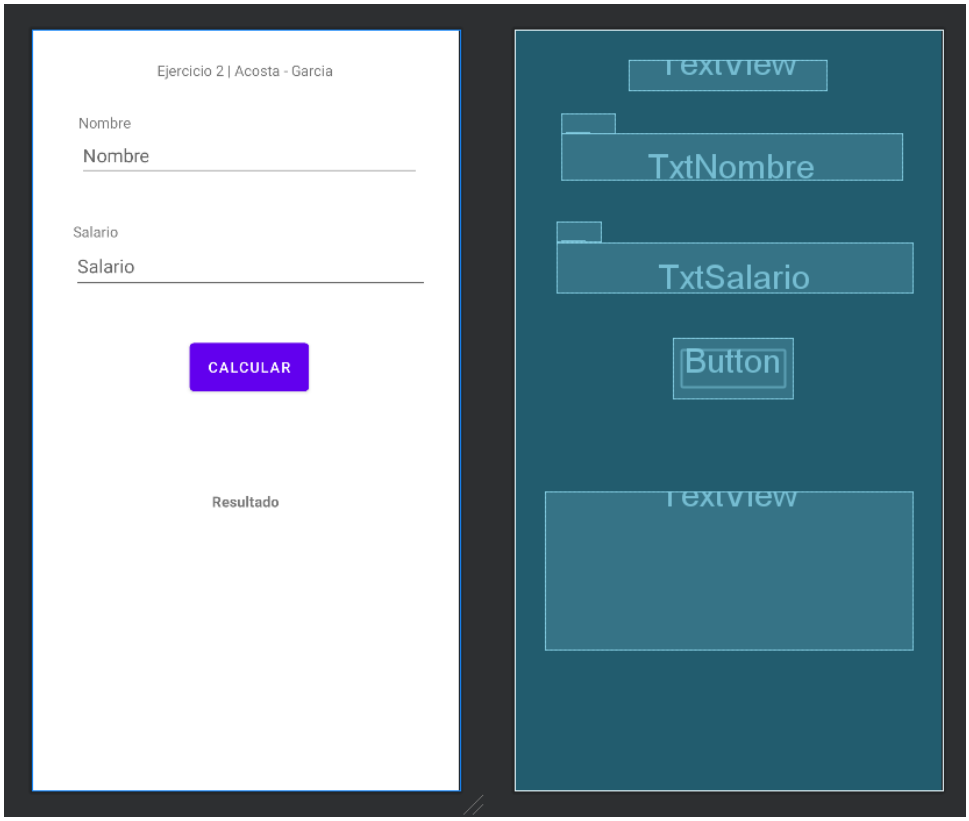

```
private var _binding: FragmentEjercicio1Binding? = null

private lateinit var btnCalcular: Button
private lateinit var txtMensaje: TextView

private lateinit var txtNota1:EditText
private lateinit var txtNota2:EditText
private lateinit var txtNota3:EditText
private lateinit var txtNota4:EditText
private lateinit var txtNota5:EditText
private lateinit var txtNombre:EditText
// This property is only valid between onCreateView and
// onDestroyView.
birdkron
private val binding get() = _binding!!

jgarcia +1
override fun onCreateView(
    inflater: LayoutInflater,
    container: ViewGroup?,
    savedInstanceState: Bundle?
): View {
    val ejercicio1ViewModel =
        ViewModelProvider( owner: this).get(Ejercicio1ViewModel::class.java)

    _binding = FragmentEjercicio1Binding.inflate(inflater, container,  attachToParent: false)
    val root: View = binding.root

    btnCalcular = binding.BtnCalcular
    txtMensaje = binding.TxtResultado

    txtNota1 = binding.TxtNota1
    txtNota2 = binding.TxtNota2
    txtNota3 = binding.TxtNota3
    txtNota4 = binding.TxtNota4
    txtNota5 = binding.TxtNota5

    txtNombre = binding.TxtNombre
    btnCalcular.setOnClickListener { realizarCalculo() }

    return root
```

```
jgarcia
private fun realizarCalculo() {
    val utils = Utils()
    val promedioNotas = PromedioNotas(
        txtNombre.text.toString(),
        utils.obtenerNumero(txtNota1),
        utils.obtenerNumero(txtNota2),
        utils.obtenerNumero(txtNota3),
        utils.obtenerNumero(txtNota4),
        utils.obtenerNumero(txtNota5)
    )

   var mensaje:String = "Estimado: " + promedioNotas.nombre
   mensaje += "\nSu promedio es de: " + promedioNotas.CalcularPromedio().toString()
   mensaje += "\nUsted ha: " + promedioNotas.MostrarAprobadoReprobado().toString()
    txtMensaje.text = mensaje
}
```

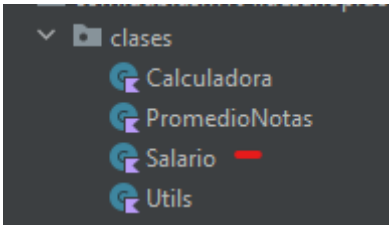
Debido al uso de Fragment pre creados por Android Studio, se hizo uso del binding, para poder acceder a los elementos de la interfaz, no como en métodos vistos anteriormente en clases, se hizo una pequeña búsqueda para ver el uso de esa parte, ya que incluía prediseñado y funcional el menú inferior usado en la solución.

Ejercicio 2 (100%)





Para la solución de este ejercicio se creó la clase de Salario donde se almacenan y estructuras los datos solicitados en el formulario, y además de realizar las operaciones, para luego solo ser utilizado en el Fragment principal, de forma similar al anterior.



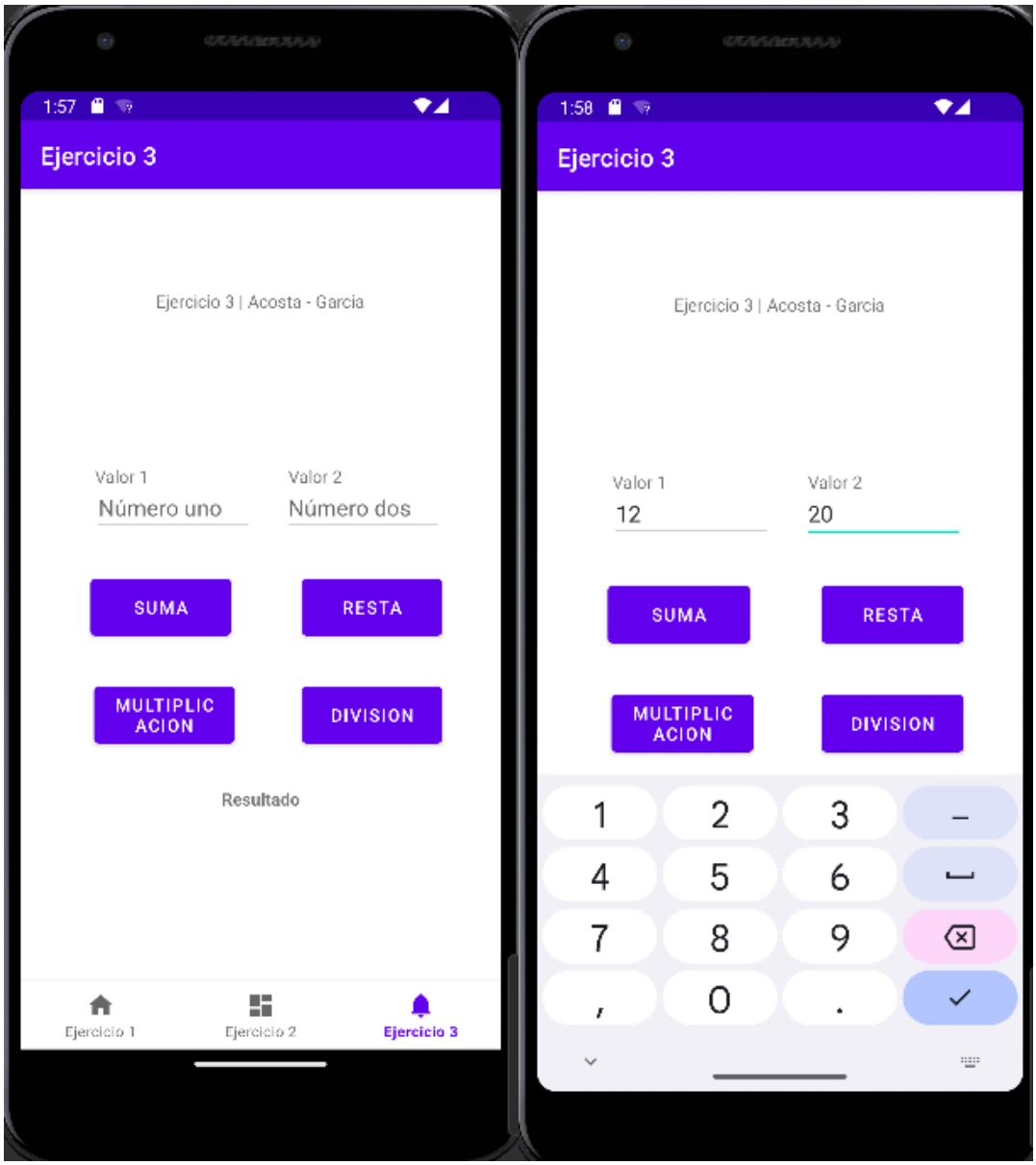
```
class Salario(
    val nombre: String,
    val salarioBase: BigDecimal
) {
    val afp = BigDecimal( val: 0.04)
    val isss = BigDecimal( val: 0.03)
    var salarioDespuesDescuentos: BigDecimal = salarioBase
    var salarioTotal = BigDecimal( val: 0)
    var renta = BigDecimal( val: 0.05)
    jgarcia
    public fun ObtenerSalarioTotal(): BigDecimal {
        return salarioTotal
    }

    jgarcia
    public fun CalcularAFP():BigDecimal{
        val retencionAfp = BigDecimal((salarioBase * afp).toString()).setScale( newScale: 2, RoundingMode.HALF_EVEN);
        salarioDespuesDescuentos -= retencionAfp
        return retencionAfp
    }

    jgarcia
    public fun CalcularISSS():BigDecimal{
        val retencionSeguro = BigDecimal((salarioBase * isss).toString()).setScale( newScale: 2, RoundingMode.HALF_EVEN);
        salarioDespuesDescuentos -= retencionSeguro
        return retencionSeguro
    }
```

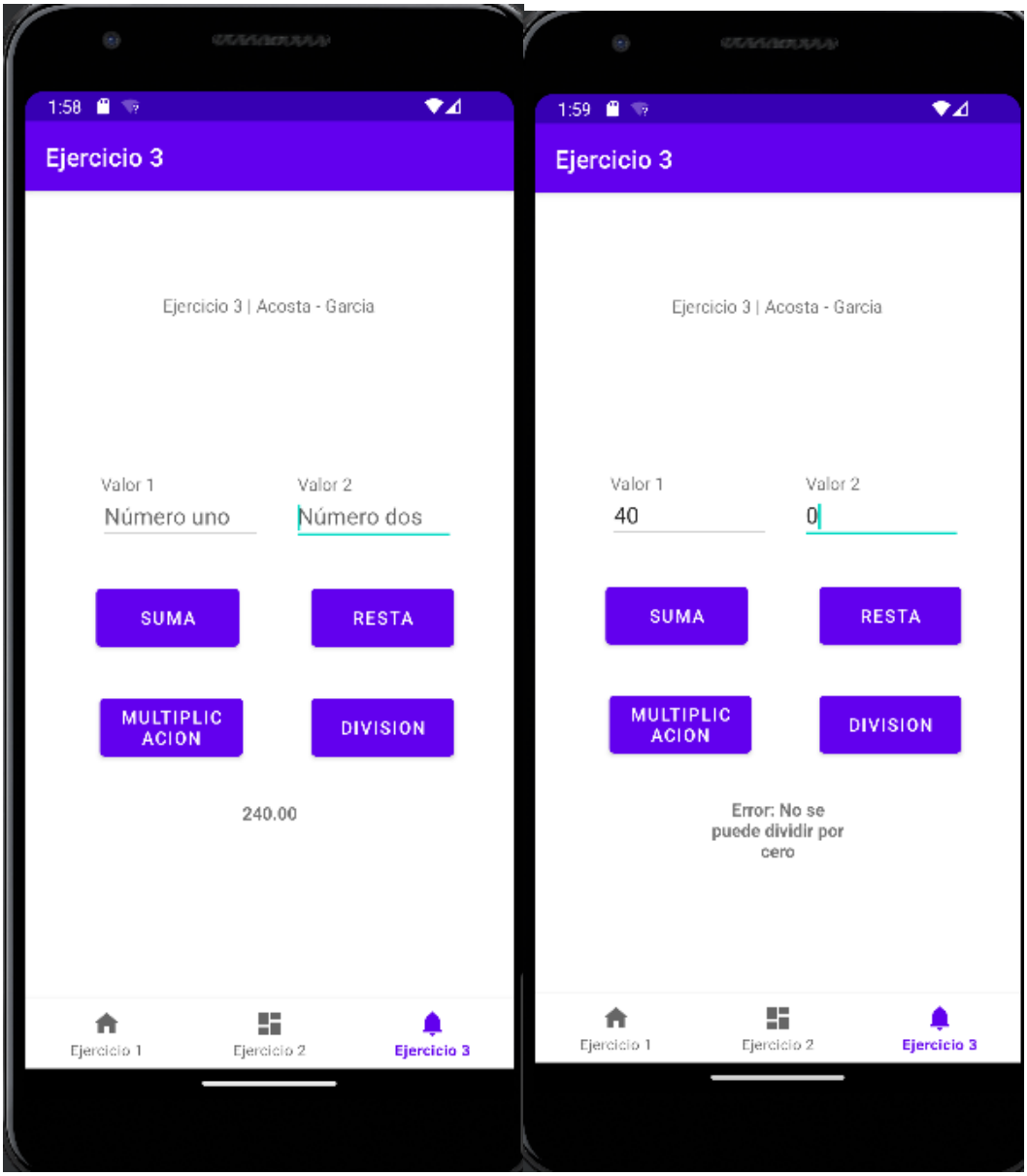
```
jgarcia
public fun CalcularRenta():BigDecimal{
    println("Salario despues de descuentos -> " + salarioDespuesDescuentos)
    if(salarioDespuesDescuentos >= BigDecimal(472.01) && salarioDespuesDescuentos <= BigDecimal(895.24)){
        renta = BigDecimal(0.1)
    }else if(salarioDespuesDescuentos >= BigDecimal(895.25) && salarioDespuesDescuentos <= BigDecimal(2038.1)){
        renta = BigDecimal(0.2)
    }else if(salarioDespuesDescuentos >= BigDecimal(2038.11)){
        renta = BigDecimal(0.3)
    }else{
        renta = BigDecimal(0)
    }
    val retencionRenta = BigDecimal((salarioDespuesDescuentos * renta).toString()).setScale( newScale: 2, RoundingMode.HALF_EVEN);
    salarioTotal = salarioDespuesDescuentos - retencionRenta
    return retencionRenta
}
```

```
class Ejercicio2Fragment : Fragment() {

    private var _binding: FragmentEjercicio2Binding? = null

    // This property is only valid between onCreateView and
    // onDestroyView.
    birdkron
    private val binding get() = _binding!!

    private lateinit var btnCalcular: Button
    private lateinit var txtMensaje: TextView

    private lateinit var txtSalario: EditText
    private lateinit var txtNombre: EditText
    jgarcia +1
    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        val ejercicio2ViewModel =
            ViewModelProvider( owner: this).get(Ejercicio2ViewModel::class.java)

        _binding = FragmentEjercicio2Binding.inflate(inflater, container,  attachToParent: false)
        val root: View = binding.root

        btnCalcular = binding.BtnCalcular
        txtMensaje = binding.TxtResultado

        txtSalario = binding.TxtSalario
        txtNombre = binding.TxtNombre

        btnCalcular.setOnClickListener { realizarCalculoRenta() }

        return root
    }
```

```
    jgarcia
    private fun realizarCalculoRenta() {
        val utils = Utils()
        val salario = Salario(
            txtNombre.text.toString(),
            utils.obtenerNumero(txtSalario),
        )

        var mensaje:String = "Estimado: " + salario.nombre
        mensaje += "\nSu Salario base es de: $" + salario.salarioBase
        mensaje += "\nSe le aplica un descuento de AFP de: $" + salario.CalcularAFP()
        mensaje += "\nSe le aplica un descuento de ISSS de: $" + salario.CalcularISSS()
        mensaje += "\nSe le aplica un descuento de Renta de: $" + salario.CalcularRenta()
        mensaje += "\nSu Salario Total es de: $" + salario.ObtenerSalarioTotal()
        txtMensaje.text = mensaje
    }

    birdkron
    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }
}
```

Ejercicio 3 (100%)





Primeramente, para crear los objetos que en este caso seran cada una de las cuatro operaciones aritméticas, creamos la clase llamada Calculadora con el siguiente codigo

```
package com.udb.dsm104.desafiopractico1.clases

import ...


 WilfredoAcosta *
class Calculadora {
     WilfredoAcosta
    fun sumar(numero1: BigDecimal, numero2: BigDecimal): BigDecimal {
        return numero1.add(numero2)
    }
     WilfredoAcosta
    fun restar(numero1: BigDecimal, numero2: BigDecimal): BigDecimal {
        return numero1.subtract(numero2)
    }
     WilfredoAcosta
    fun multiplicar(numero1: BigDecimal, numero2: BigDecimal): BigDecimal {
        return numero1.multiply(numero2)
    }
     WilfredoAcosta
    fun dividir(numero1: BigDecimal, numero2: BigDecimal): BigDecimal {
        // Validación para evitar división por cero
        if (numero2.compareTo(BigDecimal.ZERO) == 0) {
            throw ArithmeticException("No se puede dividir por cero")
        }
        return numero1.divide(numero2,  scale: 2, RoundingMode.HALF_UP)
    }
}
```

Donde las librerias de Math.BigDecimal y Math.RoundingMode nos van a ayudar a trabajar con los decimales y nos permitiran hacer la operacion de redondeo del decimal, que en nuetro caso al llegar a la mitad dentro del tercer decimal aproxima al proximo decimal (10.505 = 10.51)

```
import java.math.BigDecimal
import java.math.RoundingMode
```

Luego desarrollamos el codigo en nuestra aplicación

```kotlin
package com.udb.dsm104.desafiopractico1.ui.ejercicio3

import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.TextView
import androidx.fragment.app.Fragment
import androidx.lifecycle.ViewModelProvider
import com.udb.dsm104.desafiopractico1.databinding.FragmentEjercicio3Binding
import android.widget.Button
import android.widget.EditText
import com.udb.dsm104.desafiopractico1.clases.Calculadora
import java.math.BigDecimal

 WilfredoAcosta +2
class Ejercicio3Fragment : Fragment() {

    private lateinit var txtUno: EditText
    private lateinit var txtDos: EditText
    private lateinit var btnSuma: Button
    private lateinit var btnResta: Button
    private lateinit var btnMultiplicacion: Button
    private lateinit var btnDivision: Button
    private lateinit var txtMensaje: TextView


    //Llamamos a la clase
    private val calculadora = Calculadora()
    //Fin del llamado de clase
```

```
private var _binding: FragmentEjercicio3Binding? = null

// This property is only valid between onCreateView and
// onDestroyView.
birdkron
private val binding get() = _binding!!

WilfredoAcosta +2
override fun onCreateView(
    inflater: LayoutInflater,
    container: ViewGroup?,
    savedInstanceState: Bundle?
): View {


    val ejercicio3ViewModel =
        ViewModelProvider( owner: this).get(Ejercicio3ViewModel::class.java)

    _binding = FragmentEjercicio3Binding.inflate(inflater, container,  attachToParent: false)
    val root: View = binding.root
```

Aqui asignamos los valores de las variables creadas anteriormente a los Id’s que representa cada componente, además de eso creamos los objetos tomando en cuenta la clase hecha.

```
    txtUno = binding.TxtUno
    txtDos = binding.TxtDos
    btnSuma = binding.BtnSuma
    btnResta = binding.BtnResta
    btnMultiplicacion = binding.BtnMultiplicacion
    btnDivision = binding.BtnDivision
    txtMensaje  = binding.TxtResultado

    btnSuma.setOnClickListener { realizarOperacion(calculadora::sumar) }
    btnResta.setOnClickListener { realizarOperacion(calculadora::restar) }
    btnMultiplicacion.setOnClickListener { realizarOperacion(calculadora::multiplicar) }
    btnDivision.setOnClickListener { realizarOperacion(calculadora::dividir) }


    return root


}
```

Creamos la función de RealizarOperacion donde aplicaremos el manejo de errores, en este caso son dos errores nada más, el error por si en la division se intenta dividir entre cero y el error de ingresar números validos en dado caso alguno de los dos espacios de los valores se encuentre vacío.

```
WilfredoAcosta
private fun realizarOperacion(operacion: (BigDecimal, BigDecimal) -> BigDecimal) {
    val numero1 = obtenerNumero(txtUno)
    val numero2 = obtenerNumero(txtDos)

    if (numero1 != null && numero2 != null) {
        try {
            val resultado = operacion(numero1, numero2)
            // Mostrar el resultado con dos decimales
            mostrarResultado(resultado)
        } catch (ex: ArithmeticException) {
            // Manejar división por cero u otros errores
            mostrarMensaje( mensaje: "Error: ${ex.message}")
        }
    } else {
        mostrarMensaje( mensaje: "Por favor ingrese números válidos")
    }
}
```

Funcion de ObtenerNumero

```
WilfredoAcosta
private fun obtenerNumero(txtNumero: EditText): BigDecimal? {
    val texto = txtNumero.text.toString()
    // Validar que el texto sea un número decimal válido
    return try {
        BigDecimal(texto)
    } catch (ex: NumberFormatException) {
        null
    }
}
```

Funcion de MostrarResultado, donde al momento de que se accione el evento del boton de calcular los componentes llamados txtDos y txtUno se seteen en blanco y el txtMensaje es decir el resultado se me muestre en ese componente aplicando solo dos decimales y aproximando al decimal superior.

```
WilfredoAcosta
private fun mostrarResultado(resultado: BigDecimal) {
    //Mostrar el resultado con dos decimales
    txtDos.setText("")
    txtUno.setText("")
    txtMensaje.setText(resultado.setScale( newScale: 2, BigDecimal.ROUND_HALF_UP).toString())
}
```

Esta funcion de mostrarMensaje sera en dado caso exista algun error y que se muestre en el txtMensaje del Resultado.

```
WilfredoAcosta *
private fun mostrarMensaje(mensaje: String) {
    txtUno.setText("")
    txtDos.setText("")
    txtMensaje.text = mensaje
}
```

Como en este caso