



# **INSTITUTO TECNOLÓGICO DEL VALLE DE OAXACA**

## **DESARROLLO DE APLICACIONES MÓVILES II**



### **ACTIVIDAD Y TEMA: APLICACIÓN MÓVIL “PREDIAL” y “DOCENTES”**

ALUMNO: GUSTAVO ROMERO MONTERROZA

DOCENTE: CARDOZO JIMENEZ AMBROSIO

GRUPO: 9ª

CARRERA: Ingeniería Informática

## PROBLEMA 1 A RESOLVER: “PREDIAL”

Se desea diseñar una aplicación que permita calcular el importe total que una persona debe pagar por el impuesto predial, considerando que una persona puede tener varios predios. El costo de cada predio está en función a la zona de ubicación y para ello se cuenta con un catálogo de zonas.

Clave	Zona	Costo
MAR	Marginado	2.00
RUR	Rural	8.00
URB	Urbana	10.00
RES	Residencial	25.00

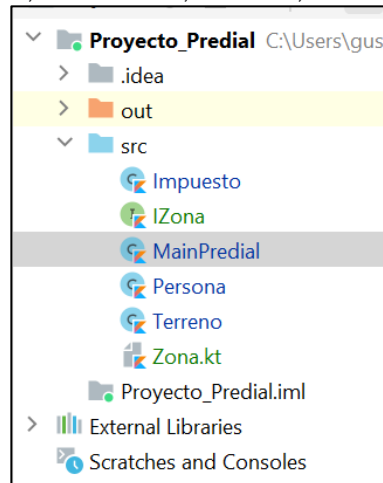
El gobierno municipal está implementando el siguiente programa de descuento:

\* Para las personas mayores o iguales de 70 años o madres solteras tiene un 70% de descuento si los pagos sean realizan en los meses de enero y febrero y de un 50% en los siguientes meses

\* Para el resto de la población hay un descuento del 40% en los meses de enero y febrero.

### PASO 1- IDENTIFICACIÓN DE LAS CLASES:

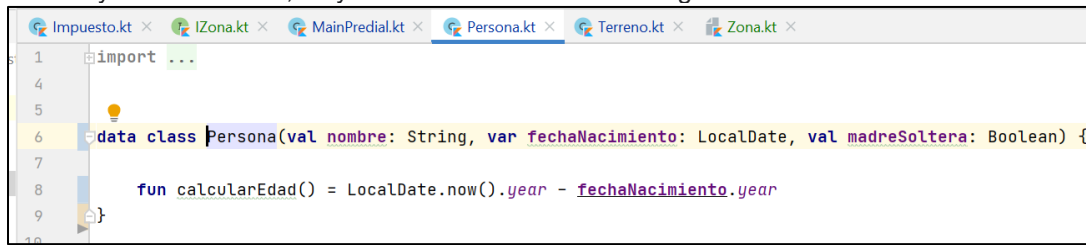
Una vez leído y analizado el problema se comenzaron a crear las clases las cuales en este caso fueron 5: Impuesto, lZona, MainPredial, Persona, Terreno y Zona (imagen 1)



(imagen 1, clases)

## 2- CLASE PERSONA

La clase persona contiene los datos solicitados en el problema, los cuales son: nombre, la fecha de nacimiento y su estado civil, tal y como se muestra en la imagen 1.1.



```
1 import ...
4
5
6 data class Persona(val nombre: String, var fechaNacimiento: LocalDate, val madreSoltera: Boolean) {
7
8     fun calcularEdad() = LocalDate.now().year - fechaNacimiento.year
9 }
10
```

(imagen 1.1, clase Persona)

### 3- CLASE IMPUESTO

En la clase impuesto como lo dice el nombre calcula el impuesto de las personas y el total que se debe pagar (imagen 1.2)

```
import java.time.LocalDate

class Impuesto(val persona: Persona, var mes: LocalDate) {
    private var PagoMes = mes.monthValue
    var arrayTerreno = arrayListOf<Terreno>()

    var arrayTerrenos = arrayListOf<Terreno>()

    fun agregarTerreno(terreno: Terreno) {
        arrayTerrenos.add(terreno)
    }

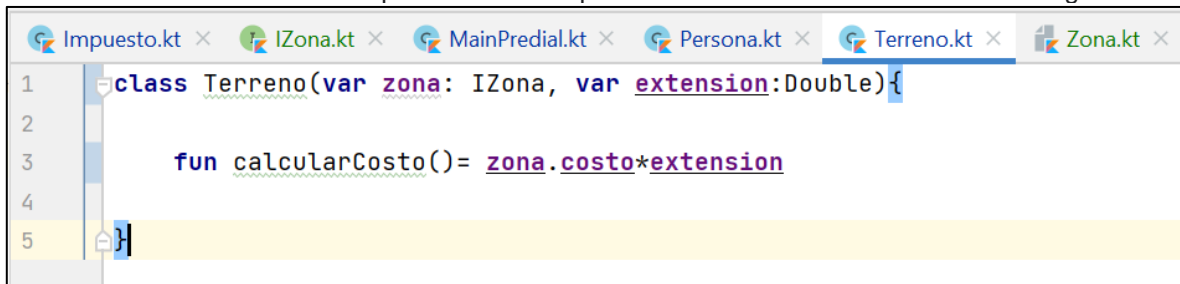
    fun calcularImpuestoTotal(): Double {
        var resultado = 0.0
        arrayTerrenos.forEach { terreno ->
            resultado += terreno.calcularCosto()
        }
        return resultado
    }

    fun calcularTotal(): Double {
        var descuento = 0.0
        if (PagoMes <= 2) {
            if (persona.calcularEdad() >= 70 || persona.madreSoltera.equals(true)) {
                descuento = 0.70
            } else {
                descuento = 0.40
            }
        } else if (persona.calcularEdad() >= 70 || persona.madreSoltera.equals(true)) {
            descuento = 0.50
        }
        var total = calcularImpuestoTotal()
        return (total - (total * descuento))
    }
}
```

(imagen 1.2)

#### 4- CLASE TERRENO

Esta clase se calcula el costo a partir de la zona por la extensión como se observa en la figura 1.4.

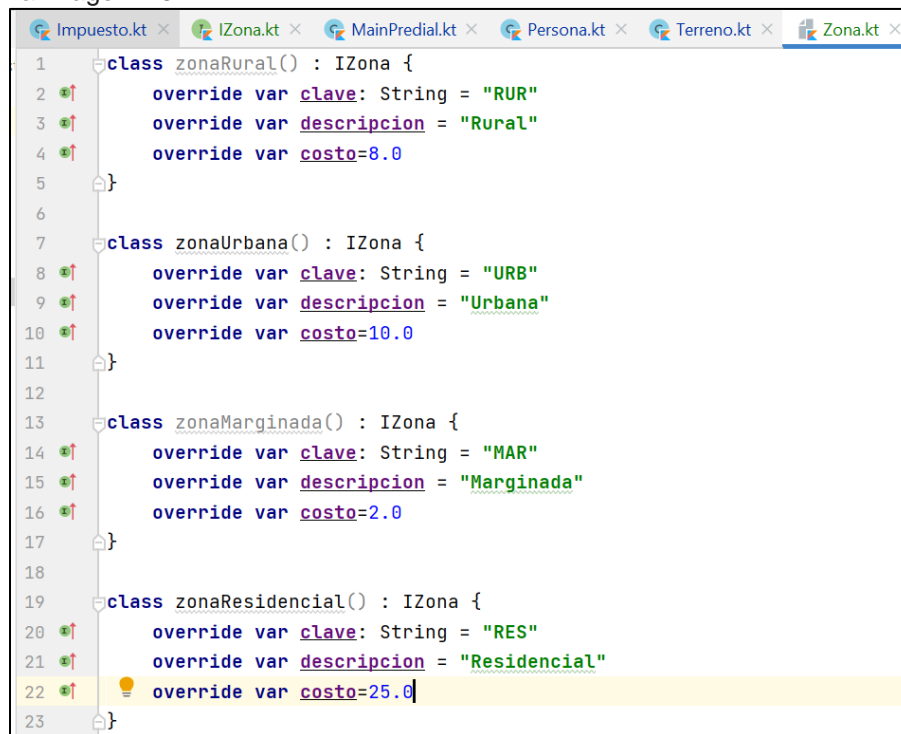


```
1 class Terreno(var zona: IZona, var extension: Double){
2
3     fun calcularCosto()= zona.costo*extension
4
5 }
```

(imagen 1.4, clase terreno)

#### 4- CLASE ZONA

Esta clase contiene los datos de cada zona como lo son el: costo, clave y la descripción, como se muestra en la imagen 1.5.



```
1 class zonaRural() : IZona {
2     override var clave: String = "RUR"
3     override var descripcion = "Rural"
4     override var costo=8.0
5 }
6
7 class zonaUrbana() : IZona {
8     override var clave: String = "URB"
9     override var descripcion = "Urbana"
10    override var costo=10.0
11 }
12
13 class zonaMarginada() : IZona {
14     override var clave: String = "MAR"
15     override var descripcion = "Marginada"
16     override var costo=2.0
17 }
18
19 class zonaResidencial() : IZona {
20     override var clave: String = "RES"
21     override var descripcion = "Residencial"
22     override var costo=25.0
23 }
```

(imagen 1.5, clase zona)

## 6- CLASE MAIN PREDIAL

En esta clase colocamos los datos que deseemos calcular (imagen 1.6) y al momento de ejecutar el programa se muestra en la parte inferior.

```
import org.junit.Test
import java.time.LocalDate

class MainPredial {
    @Test
    fun main() {

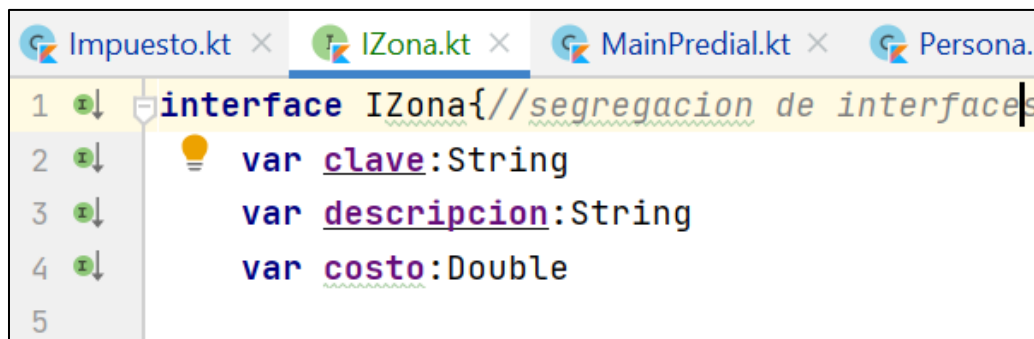
        val propietario = Persona(nombre: "Gustavo", fechaNacimiento = LocalDate.of(year: 1995, month: 3, dayOfMonth: 17), madreSoltera = false)

        val impuesto = Impuesto(propietario, mes = LocalDate.of(year: 2021, month: 1, dayOfMonth: 23))
        impuesto.agregarTerreno(Terreno(zonaResidencial(), extension = 800.0))
        impuesto.agregarTerreno(Terreno(zonaUrbana(), extension = 600.0))

        println("Total sin descuento:" + impuesto.calcularImpuestoTotal())
        println("Total :"+impuesto.calcularTotal())
    }
}
```

(imagen 1.6)

7- La clase IZona contiene los valores de clave, descripción y costo. (imagen 1.7)



```
interface IZona{ //segregacion de interfaces
    var clave:String
    var descripcion:String
    var costo:Double
}
```

(imagen 1.7)

8- RESULTADO (imagen 1.8, 1.9 y 2.0)

```
✓ Tests passed: 1 of 1 test – 31 ms  
"C:\Program Files\Android\Android Studio\jre\bin\java.exe" ...  
Total sin descuento:26000.0  
Total :26000.0  
  
Process finished with exit code 0
```

(imagen 1.8, persona que no es madre soltera y tampoco es mayor de 70 años)

```
"C:\Program Files\Android\Android Studio\jre\bin\java.exe" ...  
Total sin descuento:26000.0  
Total :13000.0  
  
Process finished with exit code 0
```

(imagen 1.9 persona mayor de 70 años)

```
"C:\Program Files\Android\Android Studio\jre\bin\java.exe" ...  
Total sin descuento:26000.0  
Total :15600.0  
  
Process finished with exit code 0
```

(imagen 2.0 persona que pago en los dos primeros meses recibe 50% de descuento)

## 8- PRINCIPIOS SOLID QUE SE CUMPLIERON

### **S: Principio de responsabilidad única**

Porque las clases, son responsables de una sola cosa.

### **L: Principio open/close**

Porque las clases usadas estan abiertas para poder extenderse y cerradas para modificarse.

## PROBLEMA 2 A RESOLVER: “CONTROL DE TRABAJADORES”

Se desea crear un programa para el control de registro de entradas y salida de personal de un centro educativo, los datos del personal son: ID, nombre completo, grado académico, (bachillerato, licenciatura, maestría o doctorado), curp, fecha de ingreso, género y clave presupuestal.

El personal tiene asignado previamente un horario de trabajo y en función a ello se va a determinar según el registro de entrada si tiene retardo o en su caso si no registro entrada podría ser una falta o permiso justificado.

El sistema cada quincena debe generar el total de inasistencias, retardos o permisos justificados de cada personal; se considera retardo si el registro se realizó en un intervalo de 11 a 20 minutos después del horario establecido, también se considera inasistencia si la asistencia se registró antes del horario de la salida establecida.

Los horarios previamente establecidos deben tener el ID del personal, día, hora de entrada y hora de salida y desde luego la fecha inicial y final que aplica ese horario (se asume que solo habrá un horario por Día)

Si la antigüedad es menor a 10 años cada 3 retardos a la quincena son contabilizados como 1 falta.

### PASO 1- IDENTIFICACIÓN DE LAS CLASES:

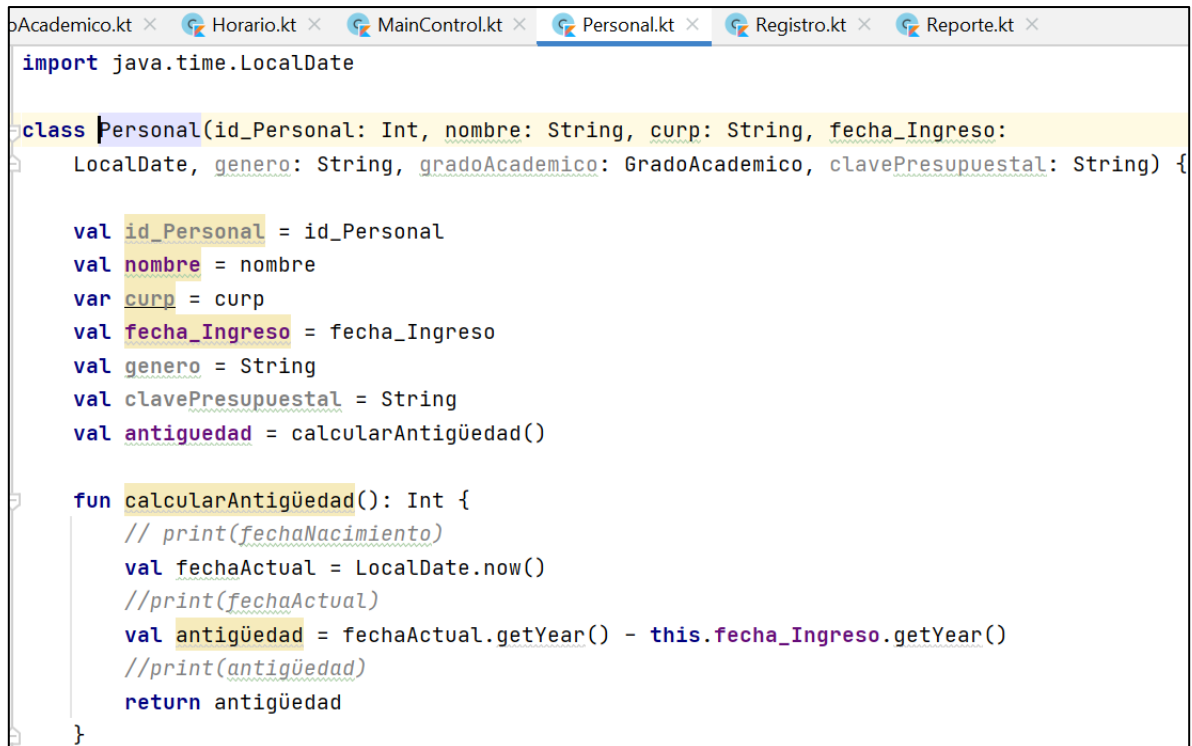
Una vez leído y analizado el problema se comenzaron a crear las clases las cuales en este caso fueron 7: Personal, GradoAcademico, Horario, Reporte, Registro, RangoHoras y MainControl (imagen 2.1)



(imagen 2.1 clases)



2- La clase personal contiene los datos que se muestran la imagen 2.0, además en esta clase se calcula la fecha de antigüedad, como se muestra en la imagen 2.2.



```
import java.time.LocalDate

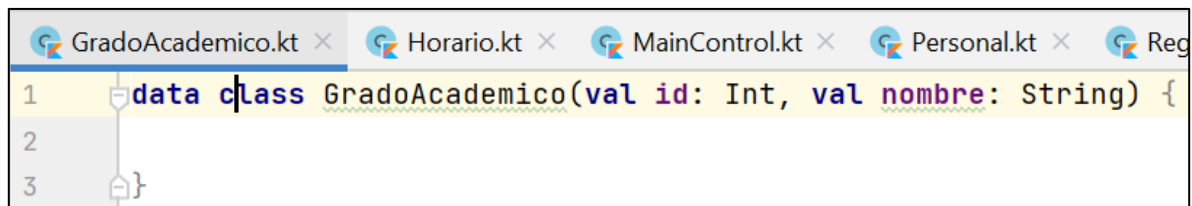
class Personal(id_Personal: Int, nombre: String, curp: String, fecha_Ingreso:
    LocalDate, genero: String, gradoAcademico: GradoAcademico, clavePresupuestal: String) {

    val id_Personal = id_Personal
    val nombre = nombre
    var curp = curp
    val fecha_Ingreso = fecha_Ingreso
    val genero = String
    val clavePresupuestal = String
    val antigüedad = calcularAntigüedad()

    fun calcularAntigüedad(): Int {
        // print(fechaNacimiento)
        val fechaActual = LocalDate.now()
        //print(fechaActual)
        val antigüedad = fechaActual.getYear() - this.fecha_Ingreso.getYear()
        //print(antigüedad)
        return antigüedad
    }
}
```

(imagen 2.2)

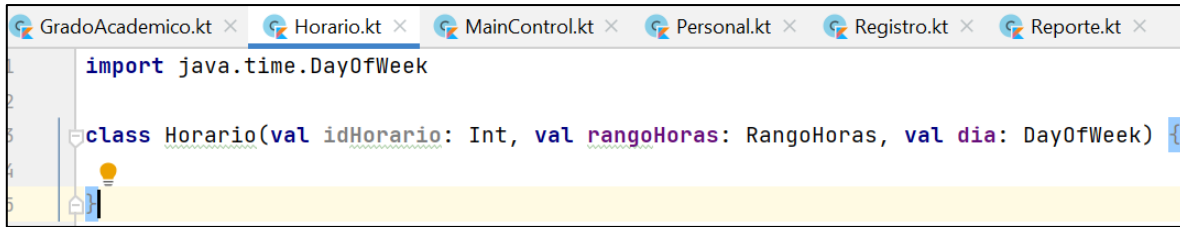
3- la clase GradoAcademico contiene el ID y el nombre, como se muestra en la imagen 2.3



```
1 data class GradoAcademico(val id: Int, val nombre: String) {
2
3 }
```

imagen 2.3

4- La clase Horario contiene los datos ID, la hora de entrada, y manda llamar la clase RangoHoras como se muestra en la imagen 2.4

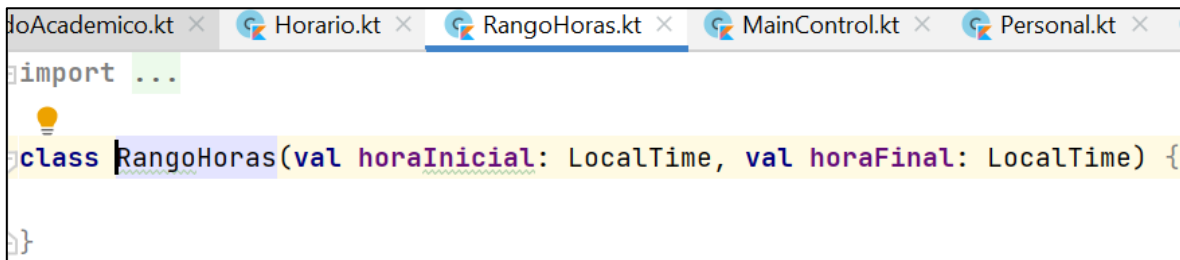


```
import java.time.DayOfWeek

class Horario(val idHorario: Int, val rangoHoras: RangoHoras, val dia: DayOfWeek) {
```

(imagen 2.4, clase horario)

5- La clase RangoHoras contiene la hora inicial y la hora final como se ve en la imagen 2.5, además estos datos se mandan llamaren la clase Horario (imagen 2.4)

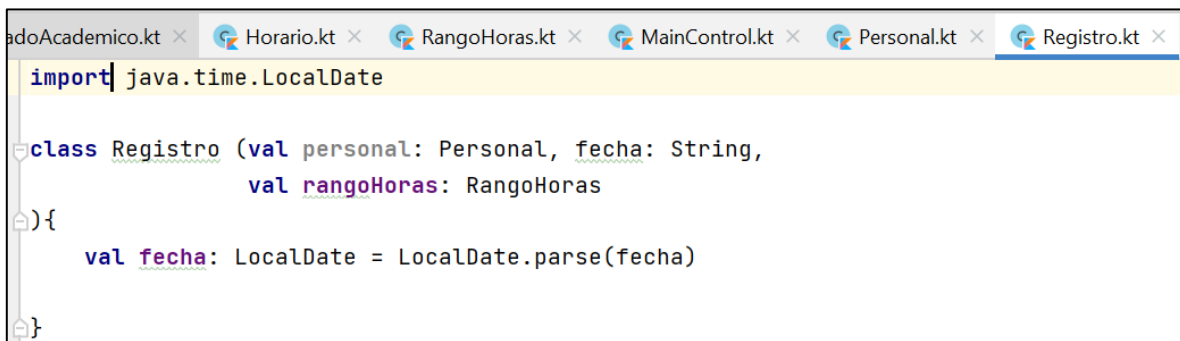


```
import ...

class RangoHoras(val horaInicial: LocalTime, val horaFinal: LocalTime) {
```

(imagen 2.5)

6- La clase Registro manda llamar a la clase personal y a la clase RangoHoras, además de contener el dato de fecha, como se observa en la imagen 2.6)



```
import java.time.LocalDate

class Registro (val personal: Personal, fecha: String,
               val rangoHoras: RangoHoras) {
    val fecha: LocalDate = LocalDate.parse(fecha)
}
```

(imagen 2.6)

7- La clase Reporte se encarga de generar los retardos (imagen 2.7), las faltas (imagen 2.8), y generar los reportes (imagen 2.9).

```
fun calcularRetardo(diaActual: LocalDate, horario: Horario): Boolean {  
    val registro = obtenerRegistro(diaActual)  
    if (registro != null) {  
        var llegada = registro.rangoHoras.horaInicial.minus(  
            horario.rangoHoras.horaInicial.getLong(ChronoField.MINUTE_OF_DAY),  
            ChronoUnit.MINUTES)  
        if (llegada.minute in 5..15){  
            retardos++  
            retardosQuincena++  
            return true  
        }  
    }  
    return false  
}
```

(imagen 2.7)

```
fun calcularFalta(diaActual: LocalDate, horario: Horario, retardo: Boolean): Boolean {  
    val registro = obtenerRegistro(diaActual)  
    if (registro == null) {  
        println("falta por ausencia")  
        faltas++  
        return true  
    }  
    if (personal.antiguedad > 10)  
        return false  
    if (registro.rangoHoras.horaFinal.isBefore(horario.rangoHoras.horaFinal) || retardo && retardosQuincena % 3 == 0) {  
        faltas++  
        return true  
    }  
    return false  
}
```

(imagen 2.8)

```

class Reporte(personal: Personal, registros: List<Registro>, horarios: List<Horario>) {

    public var retardos = 0
    public var faltas = 0
    public var permisos = 0
    val personal = personal
    private val registros = registros
    private val horarios = horarios
    private var retardosQuincena = 0

    constructor(personal: Personal, registros: List<Registro>, horarios: List<Horario>, fechaInicial: String, fechaFinal: String) {
        generarReporte(fechaInicial, fechaFinal)
    }

    fun generarReporte(fechaInicial: String, fechaFinal: String): Reporte {
        var fechaActual: LocalDate = LocalDate.parse(fechaInicial)
        var fin: LocalDate = LocalDate.parse(fechaFinal)

        //loop que se ejecuta mientras el dia actual sea menor que la fecha final
        while (!fechaActual.isAfter(fin)) {
            var diaActual = fechaActual.dayOfWeek
            if (esQuincena(fechaActual))
                reiniciarRetardosPorQuincena()
            horarios.forEach { horario ->
                if (horario.dia == diaActual) {

```

(Imagen 2.9)

8- En la clase MainControl ingreso los datos de usuario en este caso son los míos, como se muestra en la imagen 3.0, también se ingresan los datos del horario de entrada y salida (imagen 3.1)

```

MainControl.kt x Reporte.kt x
10 GradoAcademico( id: 3, nombre: "Postgrado"),
11 )
12
13 @Test
14 fun main(){
15     println("ingrese su nombre de usuario")
16     val nombre = "GUSTAVO ROMERO MONTERROZA"
17     println("ingrese su curp")
18     val curp = "ROM6990317HOCMNS08"
19     println("ingrese su fecha de ingreso")
20     val fechaI = "2021-09-18"
21     println("ingrese su género")
22     val genero = "M"
23     println("ingrese su clave presupuestal")
24     val clavePresupuestal = "984756373"
25
26
27     val personal: Personal = Personal(
28         id_Personal = 1, nombre, curp,
29         LocalDate.parse(fechaI), genero, grados[0], clavePresupuestal)

```

(imagen 3.0)

```

val personal: Personal = Personal(
    id_Personal = 1, nombre, curp,
    LocalDate.parse(fechaI), genero, grados[0], clavePresupuestal)

print("por favor asigne un horario al trabajador")

print("ingrese la hora de entrada")
val horaE = "8:00"
print("ingrese la hora de salida")
val horaS = "15:00"
println("ingrese el dia de la semana")
val dia = 1

```

(3.1, datos de horario)

9- en la clase MainControl se encuentran la lista del horario del personal por cada día de la semana, además de su registro, como se muestra en la figura 3.2 y por último se genera el reporte como lo muestra la imagen 3.3.

```

val horarios: List<Horario> = listOf(
    Horario( idHorario: 1, rangoHoras, DayOfWeek.MONDAY),
    Horario( idHorario: 2, rangoHoras, DayOfWeek.TUESDAY),
    Horario( idHorario: 3, rangoHoras, DayOfWeek.WEDNESDAY),
    Horario( idHorario: 4, rangoHoras, DayOfWeek.THURSDAY),
    Horario( idHorario: 5, rangoHoras, DayOfWeek.FRIDAY),
)
val registros: List<Registro> = listOf(
    Registro(personal, fecha: "2021-09-13", rangoHoras),
    Registro(personal, fecha: "2021-09-14", rangoHorasTarde),
    Registro(personal, fecha: "2021-09-15", rangoHoras),
    Registro(personal, fecha: "2021-09-16", rangoHoras),
    Registro(personal, fecha: "2021-09-17", rangoHoras),
    Registro(personal, fecha: "2021-09-20", rangoHoras), //1
    Registro(personal, fecha: "2021-09-21", rangoHoras),
    Registro(personal, fecha: "2021-09-22", rangoHoras),
    Registro(personal, fecha: "2021-09-23", rangoHorasTarde),
    Registro(personal, fecha: "2021-09-24", rangoHorasTarde),
    Registro(personal, fecha: "2021-09-27", rangoHorasTarde), //2
    Registro(personal, fecha: "2021-09-28", rangoHorasTarde),
    Registro(personal, fecha: "2021-09-29", rangoHorasTarde),
    Registro(personal, fecha: "2021-09-30", rangoHorasTarde),
    Registro(personal, fecha: "2021-10-01", rangoHoras),
    Registro(personal, fecha: "2021-10-02", rangoHoras),
    Registro(personal, fecha: "2021-10-03", rangoHoras),
)

```

(figura 3.2)

```

    val registroAsistencias = Reporte(personal,registros,horarios, fechaInicial: "2021-09-13", fechaFinal: "2021-10-03")
    println(registroAsistencias.personal.nombre + " ha tenido " + registroAsistencias.retardos + " retardos")
    println(registroAsistencias.personal.nombre + " ha tenido " + registroAsistencias.faltas + " faltas")
}
}

```

(imagen 3.3)

10- RESULTADOS: Al ejecutarlo (imagen 3.4) aparece de la siguiente manera:

```

✓ Tests passed: 1 of 1 test – 32 ms
"C:\Program Files\Android Studio\jre\bin\java.exe" ...
ingrese su nombre de usuario
ingrese su curp
ingrese su fecha de ingreso
ingrese su género
ingrese su clave presupuestal
por favor asigne un horario al trabajadoringrese la hora de entradaingrese la hora de salidaingrese el día de la
semana
aaaaaaaaaaaa
GUSTAVO ROMERO MONTERROZA ha tenido 7 retardos
GUSTAVO ROMERO MONTERROZA ha tenido 2 faltas
Process finished with exit code 0

```

(imagen 3.4)

## 11- PRINCIPIOS SOLID QUE SE CUMPLIERON

### **S- Principio de responsabilidad única**

Porque las clases, son responsables de una sola cosa.

### **O-Principio de Abierto/Cerrado**

Porque es capaz de extender el comportamiento de una clase, sin modificarla. O sea que las clases están abiertas para poder extenderse y cerradas para modificarse.

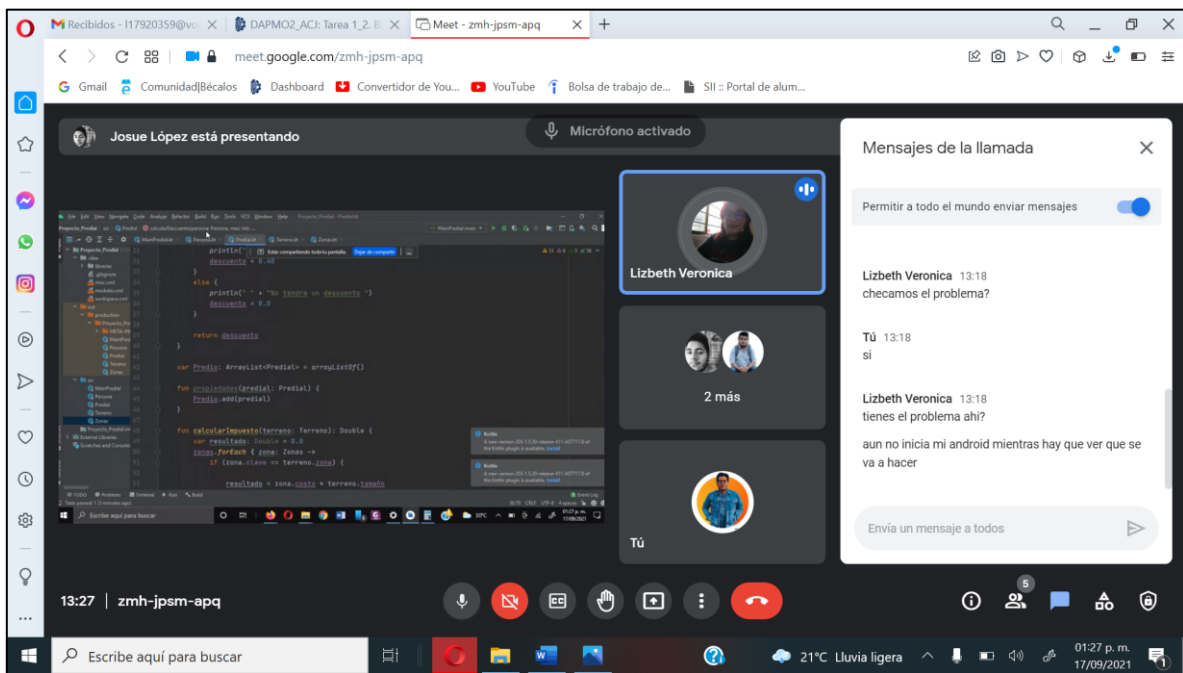
### **I-Principio de Segregación de la Interfaz**

Cada clase implemente las interfaces que va a utilizar. De este modo, agregar nuevas funcionalidades o modificar las existentes será más fácil.

# ANEXO

Para trabajar ambos proyectos me reuní con algunos de mis compañeros: (Lizbeth Verónica, Emmanuel Josué, Damián y yo [Gustavo]), ya que era complicado en ciertos puntos y nos estuvimos apoyando a través de reuniones en meet.

Adjunto evidencias:



Recibidos - 117920359@v... | INF\_DISINT\_ECO: EJERCICIO... | Meet - zmh-jpsm-apq

meet.google.com/zmh-jpsm-apq

Lizbeth Veronica está presentando

```
1 import org.junit.Test
2 import org.junit.Assert.*
3
4 class MainControl {
5
6     val personal: Personal = Personal(id=PersonalId(), nombre = "Jairo", correo="jaidm",
7     fecha_ingreso = "1966-03-13", genero="M", clavePresupuestal="")
8
9
10
11 }
```

14:47 | zmh-jpsm-apq

02:47 p.m. 17/09/2021

Ejercicio1.jar at master · jo... | Recibidos - 117920359@v... | Meet - meh-cmco-euw

meet.google.com/meh-cmco-euw

Lizbeth Veronica está presentando

```
1 fun calcularBonsento(): Double {
2     val mes = LocalDate.now().month.value
3     val descuento = 0.0
4     if (mes < 12) {
5         if (persona.calcularEdad() >= 70 || persona.adreSistema.eggsa(true)) {
8             descuento = 0.75
9         } else {
10             descuento = 0.40
11         }
12     } else if (persona.calcularEdad() >= 70 || persona.adreSistema.eggsa(true)) {
13         descuento = 0.50
14     }
15     total = calcularInventos()
16     return (total - (total * descuento))
17 }
18
19 fun calcularInventos(tarjetas: Tarjetas): Double {
20     var resultado: Double = 0.0
21     for (tarjeta in tarjetas) {
22         if (tarjeta.clave == tarjetas.zona) {
23             resultado = resultado + tarjeta.tarjeta
24         }
25     }
26     return resultado
27 }
```

21:34 | meh-cmco-euw

09:34 p.m. 22/09/2021



