

INSTITUTO TECNOLÓGICO DEL VALLE DE OAXACA

Desarrollo de Aplicaciones para Dispositivos Móviles



TEMA: PRINCIPIOS SOLID

ACTIVIDAD: BITÁCORA DE DESARROLLO DE APLICACIÓN MÓVIL

FACILITADOR: AMBROSIO CARDOSO JIMENEZ

NOMBRE DEL ESTUDIANTE: DAMIAN MARTINEZ JIMENEZ

GRADO Y GRUPO: 9 A

CARRERA: INGENIERÍA INFORMÁTICA

FECHA DE ENTREGA: 19/09/2021

APLICACIÓN 1: APP IMPORTE 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 se 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. Identificar las clases que serán creadas para desarrollar en proyecto, se identificaron 4 y una clase main en el cual se realizaron las pruebas.

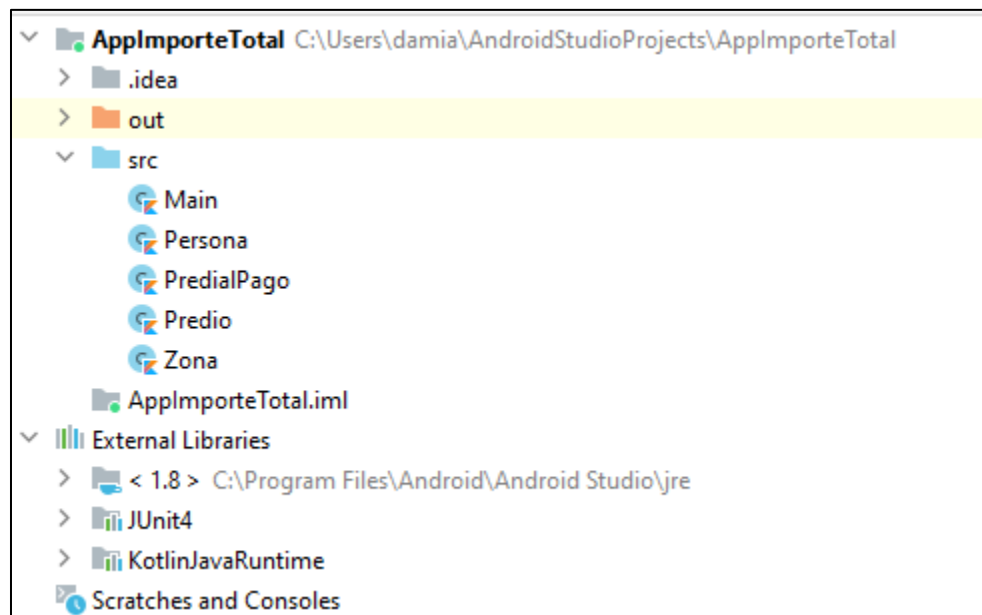
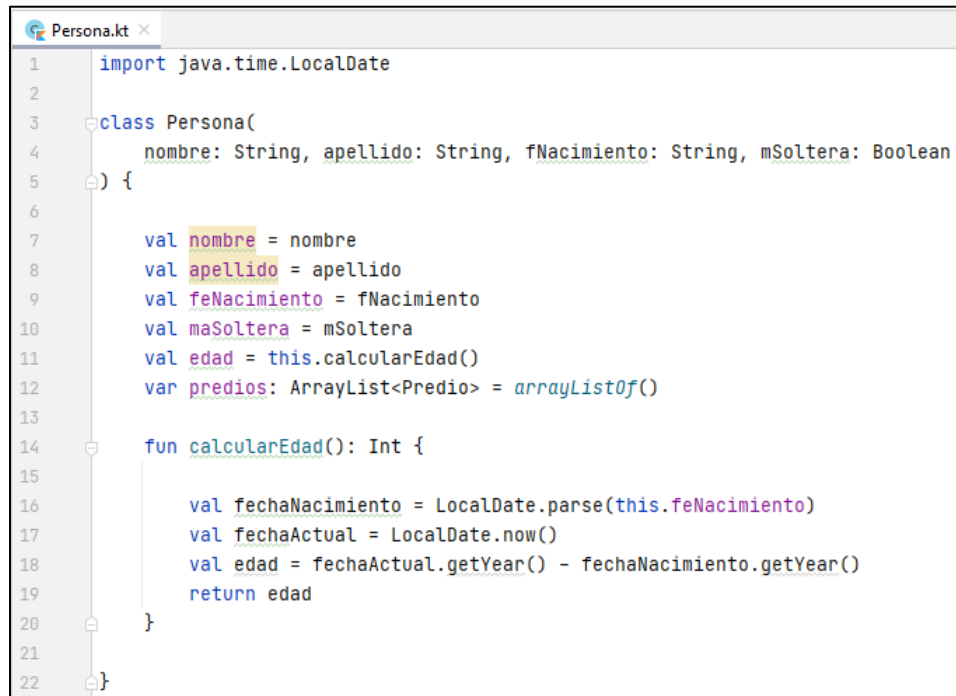


Figura 1. Clases

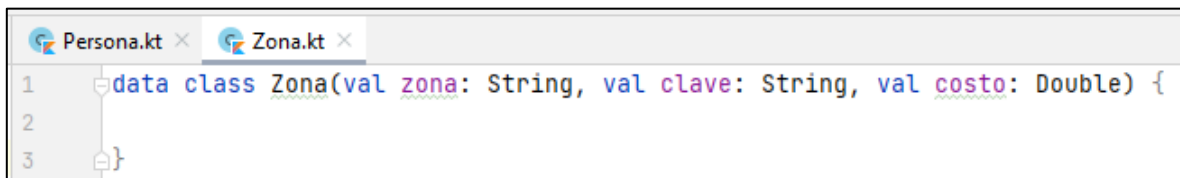
Paso 2. Codificación de la clase persona, con variables propios de una persona como nombre, apellido, fecha de nacimiento, y otros que son necesarios para resolver el problema como: si es madre soltera, edad el cual se calcula tomando en cuenta el año de nacimiento y una variable predios ya que una persona puede tener uno o más predios.



```
1 import java.time.LocalDate
2
3 class Persona(
4     nombre: String, apellido: String, fNacimiento: String, mSoltera: Boolean
5 ) {
6
7     val nombre = nombre
8     val apellido = apellido
9     val feNacimiento = fNacimiento
10    val maSoltera = mSoltera
11    val edad = this.calcularEdad()
12    var predios: ArrayList<Predio> = arrayListOf()
13
14    fun calcularEdad(): Int {
15
16        val fechaNacimiento = LocalDate.parse(this.feNacimiento)
17        val fechaActual = LocalDate.now()
18        val edad = fechaActual.getYear() - fechaNacimiento.getYear()
19        return edad
20    }
21
22 }
```

Figura 2. Clase Persona

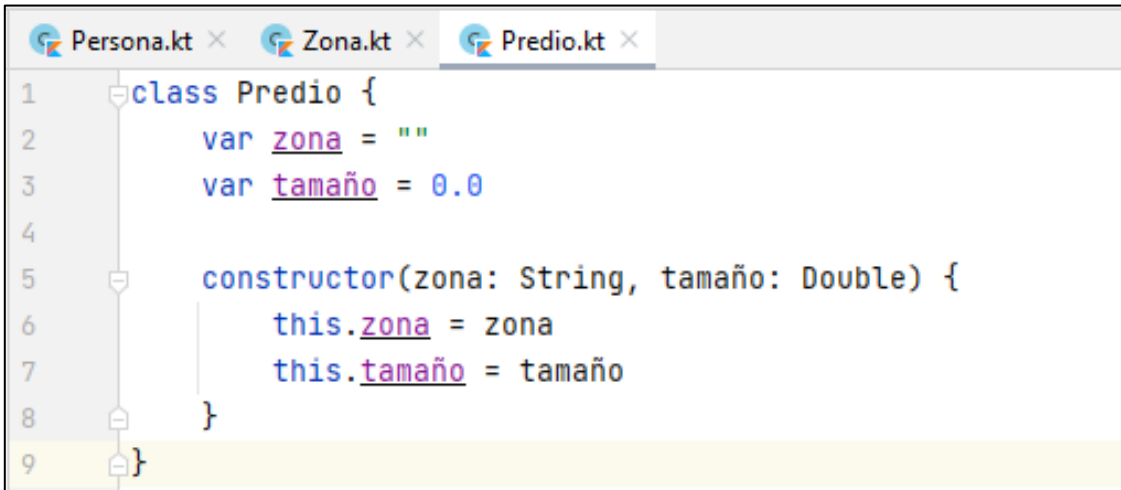
Paso 3. Codificación de una data clase Zona, contiene las variables: zona, clave y costo.



```
1 data class Zona(val zona: String, val clave: String, val costo: Double) {
2
3 }
```

Figura 3. Data clase Zona

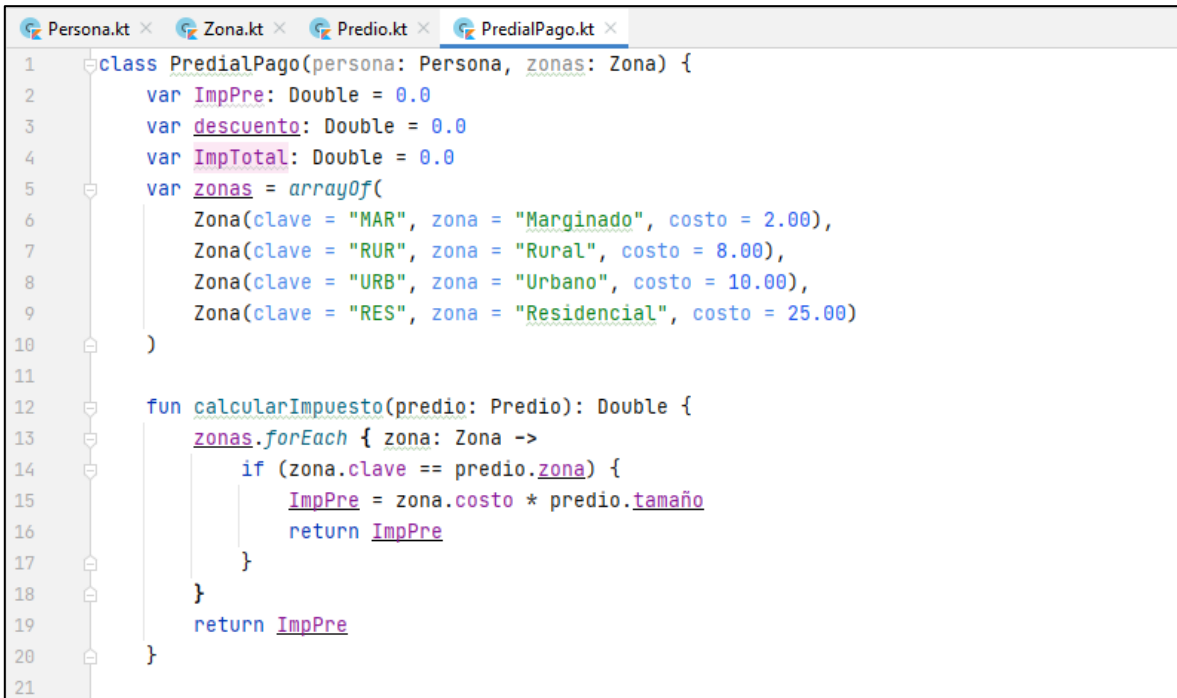
Paso 4. Codificación de la clase Predio el cual tiene como variables, zona y tamaño.



```
1 class Predio {
2     var zona = ""
3     var tamaño = 0.0
4
5     constructor(zona: String, tamaño: Double) {
6         this.zona = zona
7         this.tamaño = tamaño
8     }
9 }
```

Figura 4. Clase Predio

Paso 5. Codificación de la clase PredialPago, tiene variables, ImpuestoPredial, descuento, ImporteTotal, y un array que contiene las zonas que especifica el problema. En esta clase se realizan los cálculos, calcularImpuesto, calcularDescuento, y calcularImporteTotal.



```
1 class PredialPago(persona: Persona, zonas: Zona) {
2     var ImpPre: Double = 0.0
3     var descuento: Double = 0.0
4     var ImpTotal: Double = 0.0
5     var zonas = arrayOf(
6         Zona(clave = "MAR", zona = "Marginado", costo = 2.00),
7         Zona(clave = "RUR", zona = "Rural", costo = 8.00),
8         Zona(clave = "URB", zona = "Urbano", costo = 10.00),
9         Zona(clave = "RES", zona = "Residencial", costo = 25.00)
10    )
11
12    fun calcularImpuesto(predio: Predio): Double {
13        zonas.forEach { zona: Zona ->
14            if (zona.clave == predio.zona) {
15                ImpPre = zona.costo * predio.tamaño
16                return ImpPre
17            }
18        }
19        return ImpPre
20    }
21 }
```

Figura 5. Clase PredialPago

```

22 fun calcularDescuento(persona: Persona, mesPago: Int) {
23     var edad = persona.calcularEdad()
24     val madreSoltera = persona.maSoltera
25     if (edad >= 70 && mesPago <= 2 || madreSoltera.equals(true) && mesPago <= 2) {
26         println("Tendra un descuento de 70%")
27         descuento = ((ImpPre / 100) * 70)
28     } else if (edad >= 70 && mesPago > 2 || madreSoltera.equals(true) && mesPago > 2) {
29         println("Tendra un descuento del 50%")
30         descuento = ((ImpPre / 100) * 50)
31     } else if (mesPago <= 2) {
32         println("Tendra un descuento del 40%")
33         descuento = ((ImpPre / 100) * 40)
34     } else {
35         println("No tendra un descuento ")
36         descuento = 0.0
37     }
38 }
39 }
40
41 fun calcImpuestoTotal(predio: Predio): Double {
42     ImpTotal = ImpPre - descuento
43
44     return ImpTotal
45 }
46
47 }

```

Figura 6. Continuación clase PredialPago

Paso 6. Codificación de la clase Main, en el cual se realizaron pruebas, haciendo uso de la librería Junit4

```

1 import org.junit.Test
2
3 class Main {
4     @Test
5     fun main() {
6         val persona: Persona =
7             Persona(
8                 nombre = "Damian",
9                 apellido = "Martinez",
10                 fNacimiento = "1998-09-10",
11                 mSoltera = false
12             )
13
14         val zona: Zona = Zona(clave = "URB", zona = "Urbano", costo = 10.00)
15         val predial: PredialPago = PredialPago(persona = persona, zonas = zona)
16
17         println("Nombre: +(persona.nombre + " ") + (persona.apellido))
18         println("Edad: " + persona.edad)
19         persona.predios.add(Predio(zona = "URB", tamaño = 400.0))
20         println("El impuesto predial es: $" + predial.calcularImpuesto(persona.predios.get(0)) + "\n")
21         predial.calcularDescuento(persona = persona, mesPago = 5)
22         println("El descuento es de: $" + predial.descuento + "\n")
23         println("El impuesto predial total es: $" + predial.calcImpuestoTotal(persona.predios.get(0)))
24     }
25 }

```

Figura 7. Clase Main para pruebas

Resultados

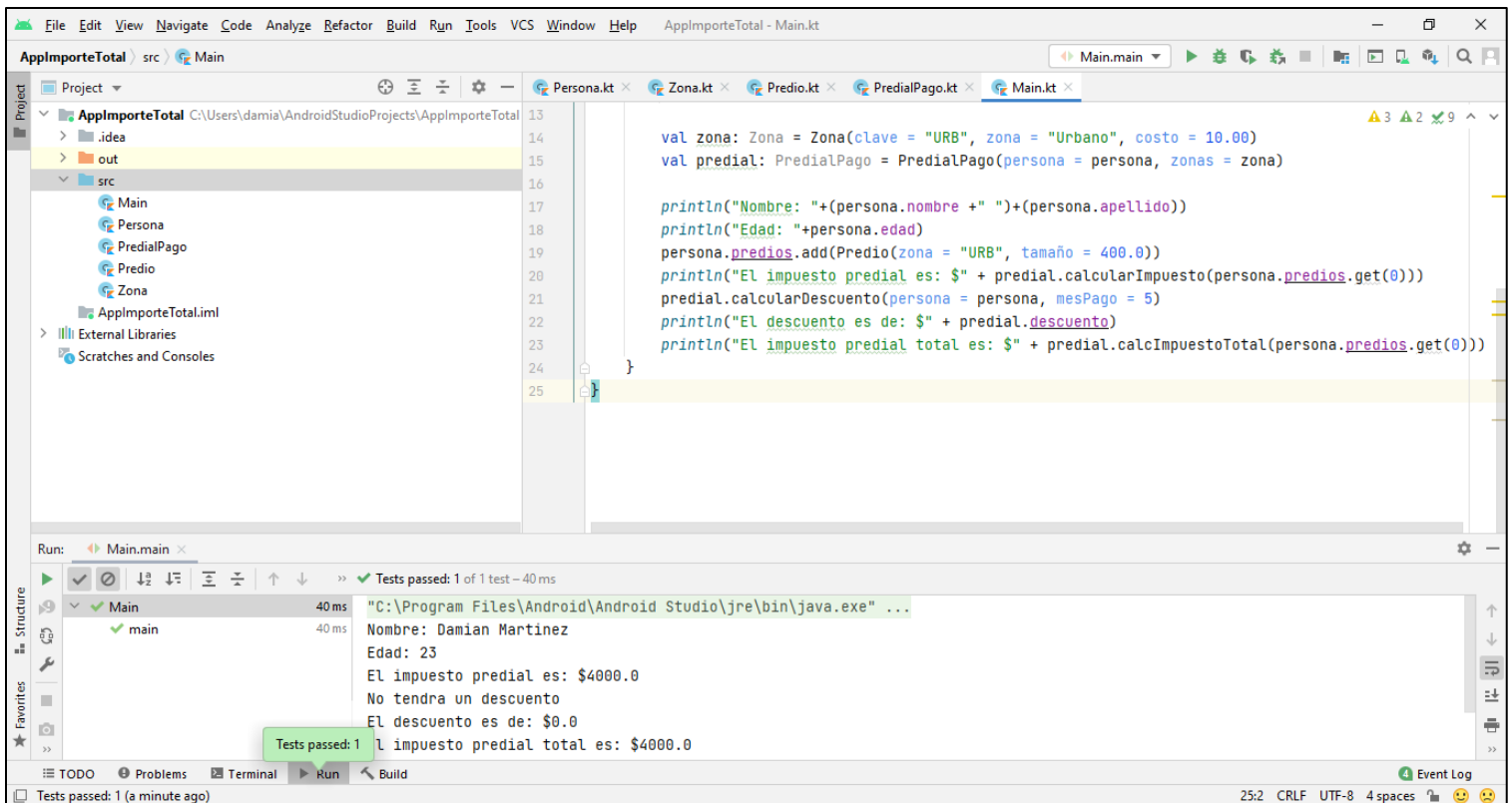


Figura 8. Prueba 1

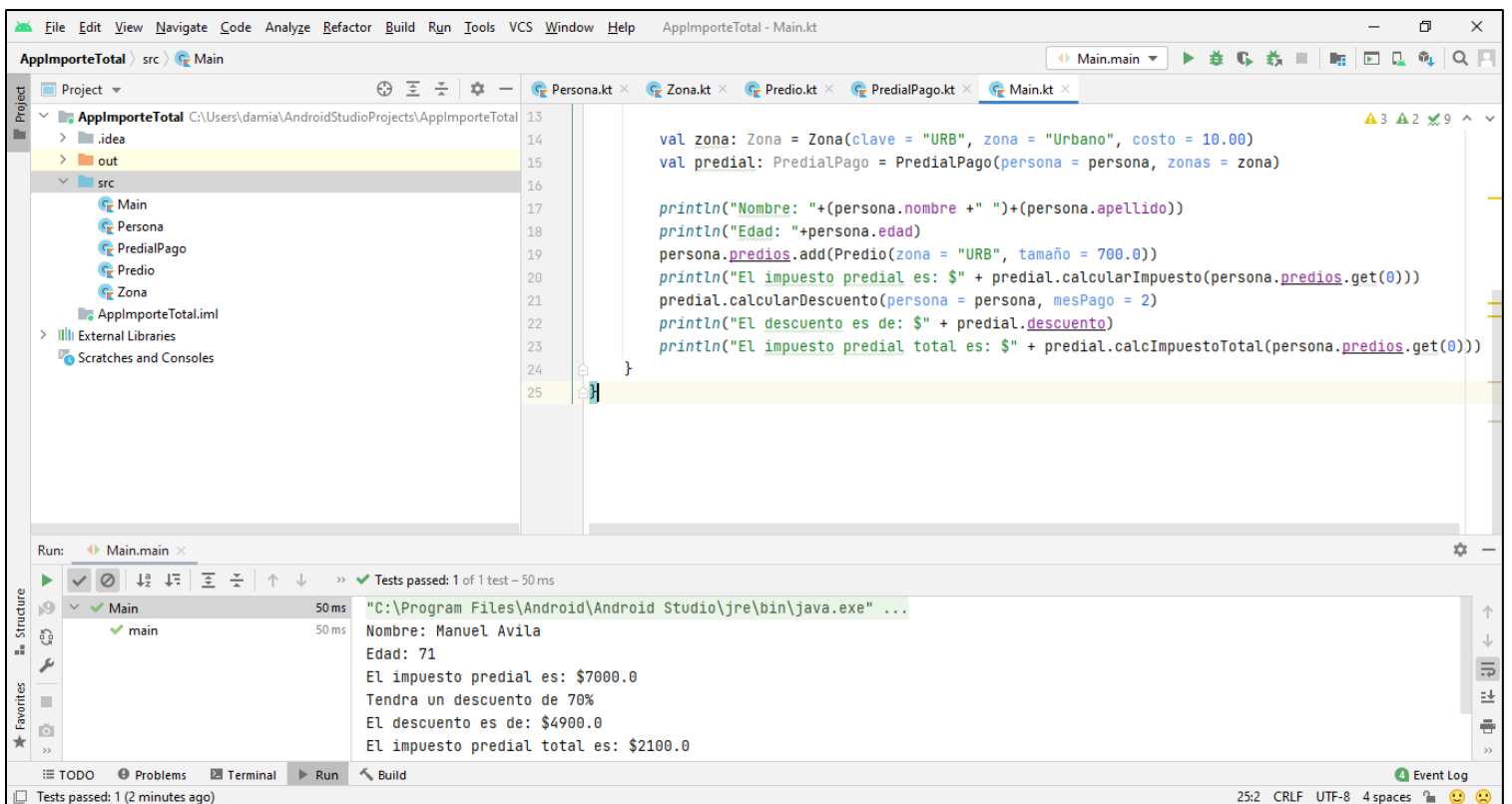


Figura 9. Prueba 2

Principios SOLID que se cumplen.

- Principio de responsabilidad única (S):
Cada clase es responsable de realizar una única cosa.
- Principio Open/Closed
La clase PredialPago es la única que se encarga de realizar cálculos, se puede decir que la clase está abierta a nuevas funcionalidades de cálculo, sin afectar el código que ya existe.

Url del repositorio: <https://github.com/damian-9/AppImportePredial>

APLICACIÓN 2: CONTROL TRABAJADORES

```
13 2. Se desea crear un programa para el control de registro de entrada y salida de personal de un centro educativo
14 los datos del personal son: identificador, nombre completo, grado academico (Bachillerato, licenciatura, maestría o
15 doctorado), curp
16 fecha de ingreso, genero y clave presupuestal. El personal tiene asignado previamente un horario de trabajo y en
17 función a ello
18 se va a determinar según el registro de entrada si tiene retardo o en su caso sino registró entrada podria ser una
19 falta o permiso justificado
20 El sistema cada quincena debe generar el total de inasistencias, retardos o permisos justificados de cada personal;
21 se considera retardo
22 si el registro se realizó en un intervalo entre 11 a 20 minutos después del horario establecido. También se
23 considera inasistencia si la salida se registró antes del horario de salida establecida
24 Los horarios previamente establecidos debe tener el id del personal, día, hora entrada y hora 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 serán contabilizados como 1 falta.
```

Figura 1. Problema a resolver

Paso 1. Se identifican las clases, en este caso, GradoAcademico, Horario, Personal, Registro, Reporte, y una clase MainControl que ayudara a realizar las pruebas.

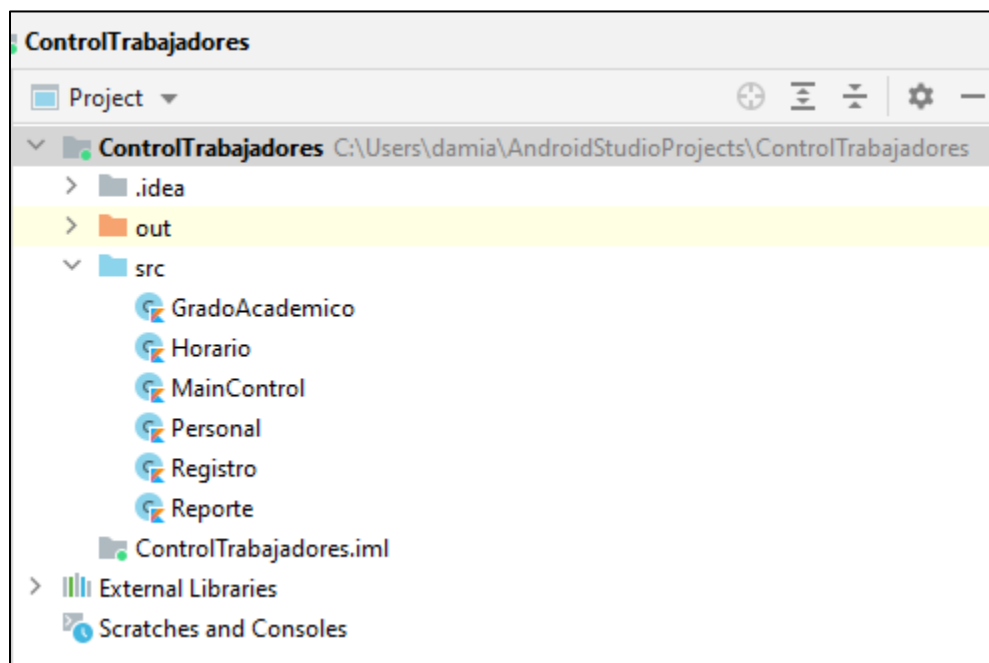
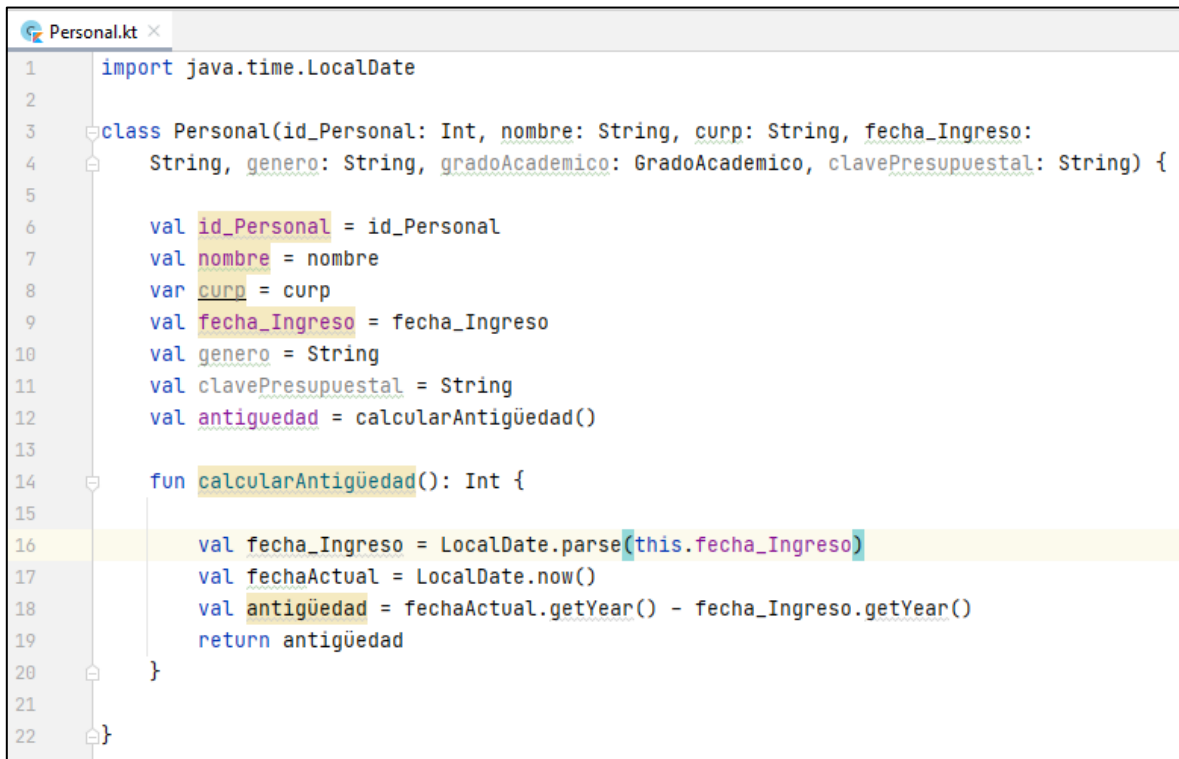


Figura 2. Clases

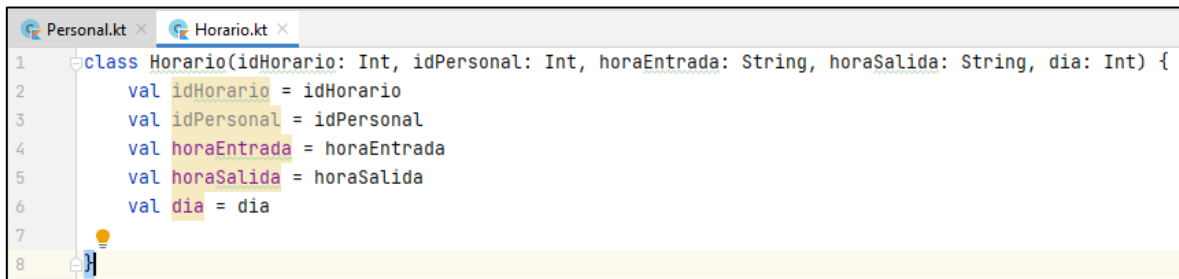
Paso 2. Se crea la clase Personal, con sus respectivas variables.



```
1 import java.time.LocalDate
2
3 class Personal(id_Personal: Int, nombre: String, curp: String, fecha_Ingreso:
4     String, genero: String, gradoAcademico: GradoAcademico, clavePresupuestal: String) {
5
6     val id_Personal = id_Personal
7     val nombre = nombre
8     var curp = curp
9     val fecha_Ingreso = fecha_Ingreso
10    val genero = String
11    val clavePresupuestal = String
12    val antigüedad = calcularAntigüedad()
13
14    fun calcularAntigüedad(): Int {
15
16        val fecha_Ingreso = LocalDate.parse(this.fecha_Ingreso)
17        val fechaActual = LocalDate.now()
18        val antigüedad = fechaActual.getYear() - fecha_Ingreso.getYear()
19        return antigüedad
20    }
21
22 }
```

Figura 3. Clase Personal

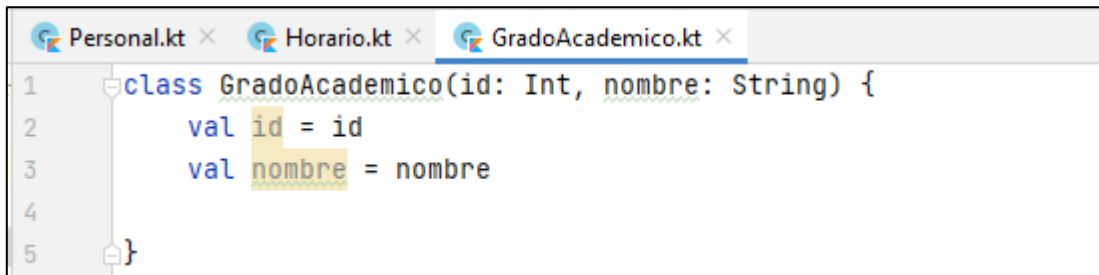
Paso 3. Se crea la Clase, que contiene como atributos un idHorario, el idPersonal, horaEntrada, horaSalida y día.



```
1 class Horario(idHorario: Int, idPersonal: Int, horaEntrada: String, horaSalida: String, dia: Int) {
2     val idHorario = idHorario
3     val idPersonal = idPersonal
4     val horaEntrada = horaEntrada
5     val horaSalida = horaSalida
6     val dia = dia
7
8 }
```

Figura 4. Clase Horario

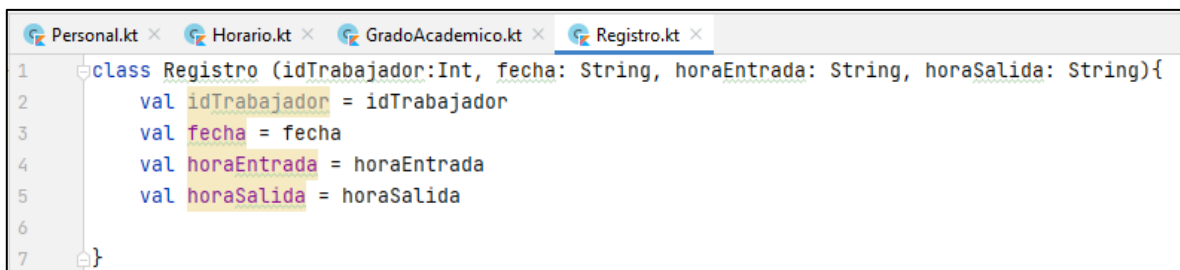
Paso 4. Clase GradoAcademico, con variables id y nombre.



```
1 class GradoAcademico(id: Int, nombre: String) {
2     val id = id
3     val nombre = nombre
4
5 }
```

Figura 5. Clase GradoAcademico

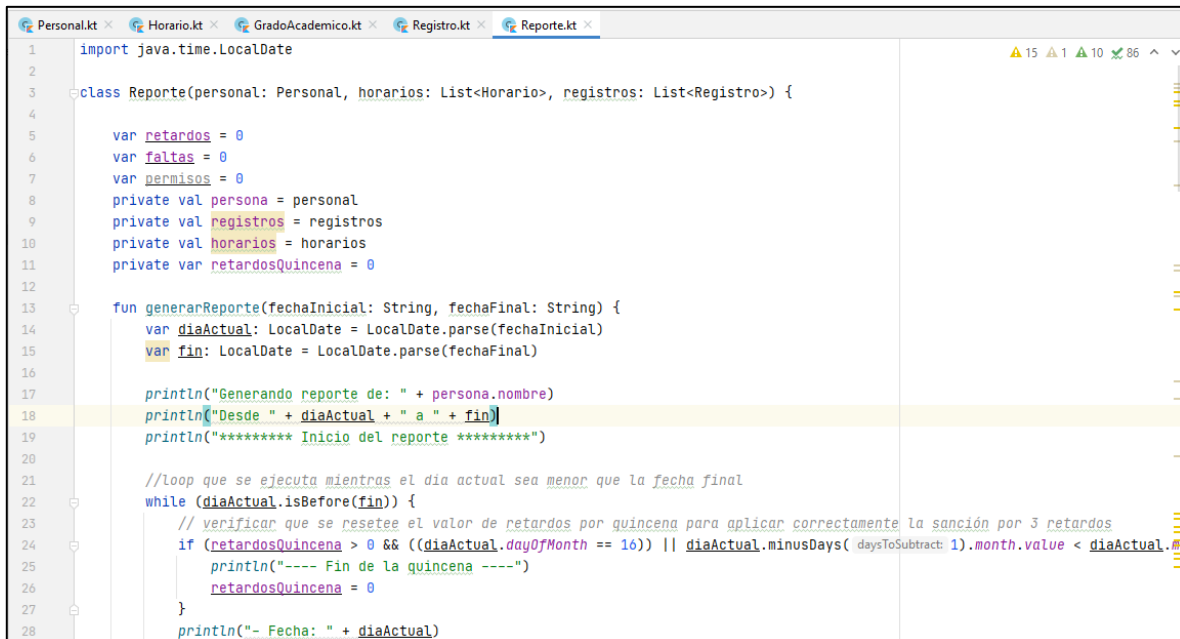
Paso 5. Clase Registro, con variables idTrabajador, fecha, horaEntrada y horaSalida.



```
1 class Registro(idTrabajador: Int, fecha: String, horaEntrada: String, horaSalida: String) {
2     val idTrabajador = idTrabajador
3     val fecha = fecha
4     val horaEntrada = horaEntrada
5     val horaSalida = horaSalida
6
7 }
```

Figura 6. Clase Registro

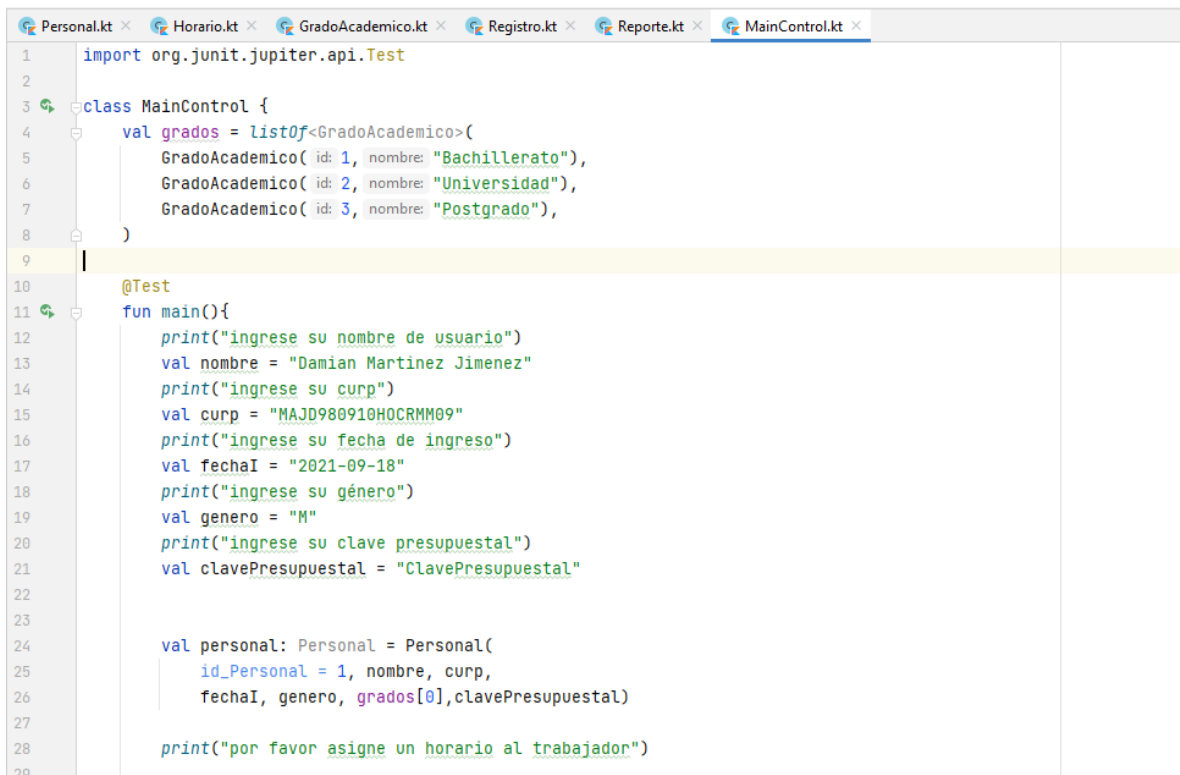
Paso 6. Codificación de la clase Reporte, el cual se encargará de generar el reporte, verificar el horario, calcular retardos y faltas.



```
1 import java.time.LocalDate
2
3 class Reporte(personal: Personal, horarios: List<Horario>, registros: List<Registro>) {
4
5     var retardos = 0
6     var faltas = 0
7     var permisos = 0
8     private val persona = personal
9     private val registros = registros
10    private val horarios = horarios
11    private var retardosQuincena = 0
12
13    fun generarReporte(fechaInicial: String, fechaFinal: String) {
14        var diaActual: LocalDate = LocalDate.parse(fechaInicial)
15        var fin: LocalDate = LocalDate.parse(fechaFinal)
16
17        println("Generando reporte de: " + persona.nombre)
18        println("Desde " + diaActual + " a " + fin)
19        println("***** Inicio del reporte *****")
20
21        //loop que se ejecuta mientras el dia actual sea menor que la fecha final
22        while (diaActual.isBefore(fin)) {
23            // verificar que se resetee el valor de retardos por quincena para aplicar correctamente la sanción por 3 retardos
24            if (retardosQuincena > 0 && ((diaActual.dayOfMonth == 16) || diaActual.minusDays(daysToSubtract: 1).month.value < diaActual.month.value)) {
25                println("---- Fin de la quincena ----")
26                retardosQuincena = 0
27            }
28            println("- Fecha: " + diaActual)
29        }
30    }
31 }
```

Figura 7. Clase Reporte

Paso 7. Codificación de la clase Main, con el cual se realizarán los tests.



```
1 import org.junit.jupiter.api.Test
2
3 class MainControl {
4     val grados = listOf<GradoAcademico>(
5         GradoAcademico( id: 1, nombre: "Bachillerato"),
6         GradoAcademico( id: 2, nombre: "Universidad"),
7         GradoAcademico( id: 3, nombre: "Postgrado"),
8     )
9
10    @Test
11    fun main(){
12        print("ingrese su nombre de usuario")
13        val nombre = "Damian Martinez Jimenez"
14        print("ingrese su curp")
15        val curp = "MAJD980910HOCRRM09"
16        print("ingrese su fecha de ingreso")
17        val fechaI = "2021-09-18"
18        print("ingrese su género")
19        val genero = "M"
20        print("ingrese su clave presupuestal")
21        val clavePresupuestal = "ClavePresupuestal"
22
23
24        val personal: Personal = Personal(
25            id_Personal = 1, nombre, curp,
26            fechaI, genero, grados[0],clavePresupuestal)
27
28        print("por favor asigne un horario al trabajador")
29    }
```

Figura 8. Clase MainControl

Resultados

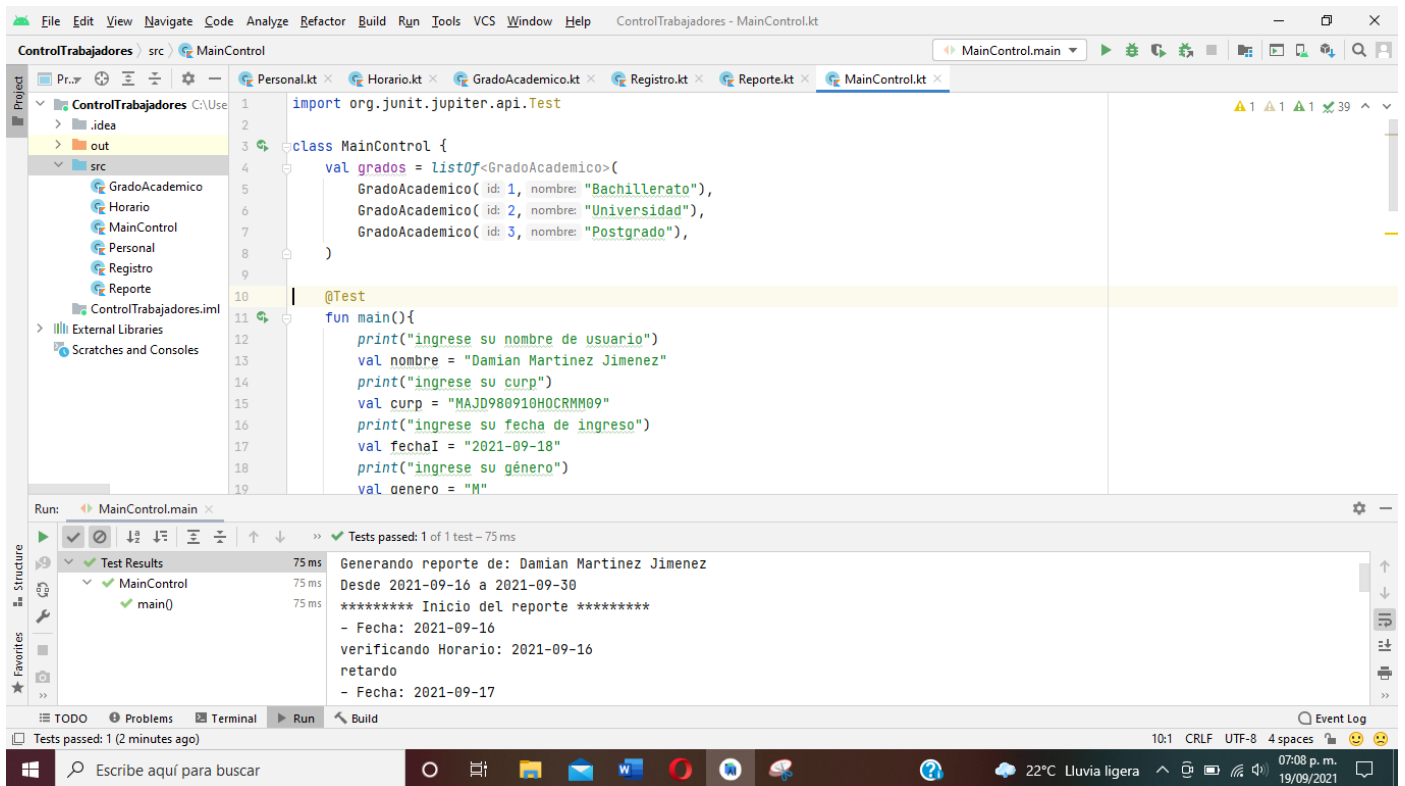


Figura 9. Prueba

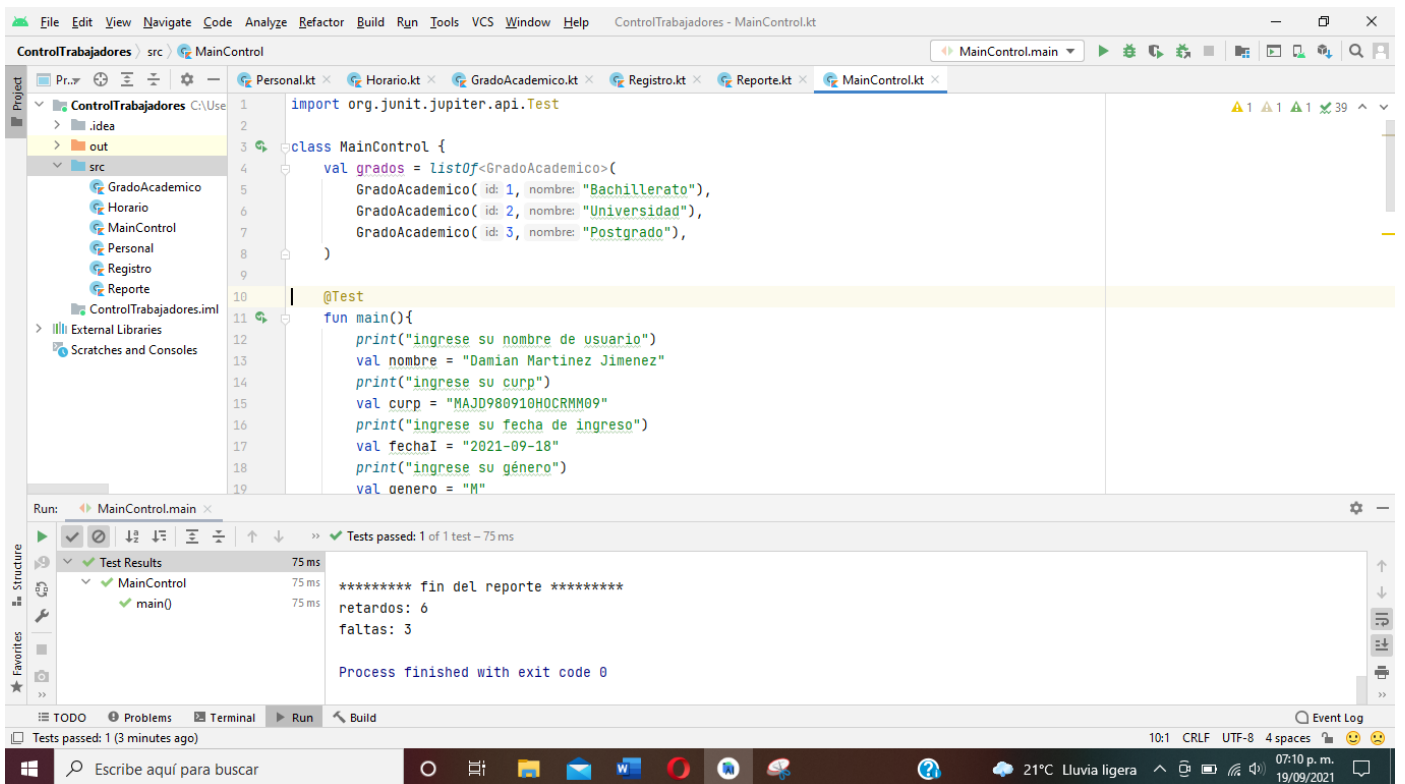


Figura 10. Continuación prueba

Principios SOLID que se cumplen.

- Principio de responsabilidad única (S):
Cada clase es responsable de realizar una única cosa.
- Principio Open/Closed
La clase Reporte es la única que se encarga de realizar cálculos, como el de generar el reporte, calcular faltas y retardos, se puede decir que la clase está abierta a nuevas funcionalidades de cálculo, sin afectar el código que ya existe.
- Principio de segregación de interfaces
Ninguna clase depende de métodos que no usa, lo cual ayuda reutilizar el código.

Url del repositorio:

<https://github.com/damian-9/controlTrabajadores/tree/main/ControlTrabajadores>

Nota: Para desarrollar estas aplicaciones me reuní con algunos de mis compañeros (Lizbeth Verónica, Emmanuel Josué, Gustavo y yo (Damian)), ya que era complicado en ciertos puntos y entre todos nos estuvimos apoyando a través de reuniones en meet.