Entrega Final

Nombres: Valentina San Martín, Marlene Lagos y Matías Mujica Alevropulos

Fecha de Entrega: 4 de Julio de 2021

Repositorio GitHub de Versiones de Proyecto

Distritos Electorales de Chile (28), Wikipedia

EP1.1. Realizar un análisis de los datos a utilizar y principales funcionalidades a implementar que dan sentido a la realización del proyecto.

El proyecto a realizar tratará acerca de los casos que manejan los fiscales dentro de la Fiscalía de Chile. El objetivo del programa será brindar al Ministerio Público una forma de mantener organizadas las causas y los fiscales que tienen a su disposición a lo largo de todo el país.

Los datos a utilizar son:

-Causas que posee el Ministerio Público, incluyendo el código que las identifica, los datos del fiscal a cargo del caso, la información sobre los procedimientos realizados en el caso,el estado del caso y el lugar en donde se desarrolla.

-Los fiscales que trabajan para el Ministerio Público, incluyendo sus nombres, rut, su especialidad en el ámbito jurídico, su ubicación actual y los casos que tiene a su cargo.

Las funcionalidades a implementar son:

1.- Mostrar fiscales: Muestra por pantalla la información de todos los fiscales dentro de la fiscalía.

2.-Mostrar Causas: Muestra por pantalla la información de todas las causas dentro de la fiscalía, incluyendo una lista de sus procedimientos.

3.-Buscar Fiscal: El usuario ingresa el rut del fiscal y el programa le muestra la información de ese fiscal, incluyendo la información sobre sus causas y los procedimientos de las mismas.

4.-Buscar causa: El usuario ingresa el código de la causa y el programa le muestra la información de esa causa, incluyendo la información del Fiscal a cargo y sus procedimientos.

5.-Nuevo fiscal: Se le pide al usuario ingresar la información del fiscal y a partir de eso ingresar el nuevo fiscal al sistema.

6.-Nueva causa: Se le pide al usuario ingresar código de causa nueva, su estado (abierto,cerrado), tipo, su distrito, creará una causa a nuestro mapa.

7.-Agregar procedimiento a una causa: Se pide al usuario o fiscal ingresar código de una causa existente, se ingresa nombre de peritaje, nombre del participante, su rol, si desea ingresar mas participantes y el resultado de su peritaje.

8.-Asignar una causa a un fiscal: Primero se pregunta qué causa (sin Fiscal asignado) quieres asignar, luego entregamos una sugerencia de Fiscales recomendados para la causa en específico (Basado en distrito y especialidad)

9.-Modificar el distrito de un fiscal: A partir del RUT de nuestro fiscal, se asignará un nuevo distrito basado en la elección del usuario.

10.-Cambiar el fiscal encargado de una causa: Se le pide al usuario ingresar código de causa y el RUT del nuevo fiscal encargado de tal causa.

11.-Cambiar el resultado de un procedimiento: Se le pide al usuario ingresar código de causa, se muestra causa por pantalla, se muestran procedimientos, se le pide al usuario el número de procedimiento a cambiar resultado.

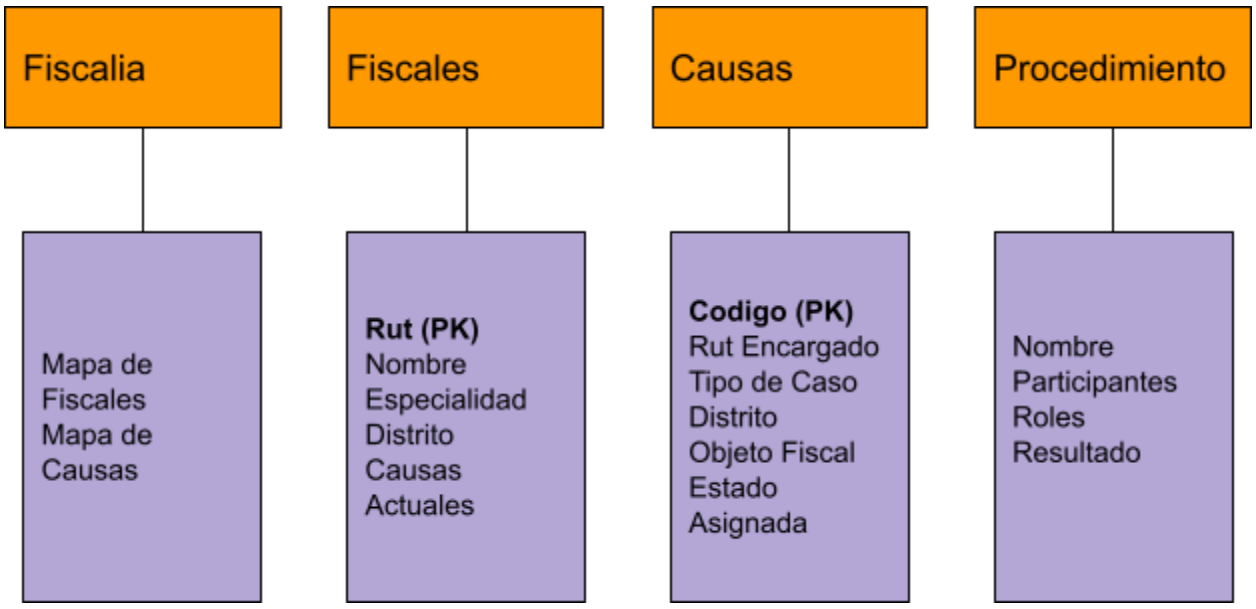
12.-Eliminar Fiscal: Se elimina fiscal a partir de su RUT.

13.-Eliminar Causa: Se elimina la causa a partir de su Código.

14.-Eliminar Procedimiento: Se elimina el procedimiento a partir de su Código

Errores menores: Errores de excepción en caso de resultado no esperado, algunos errores menores de formato.

EP1.2. Diseño conceptual de clases del Dominio y su código en Java



Las clases a implementar en el programa son:

1. Procedimiento: Esta clase guardará la información de los diferentes procedimientos que se vayan realizando en sus respectivos casos. Sus atributos son:

a) String nombreProc: Guarda el nombre del procedimiento, o sea si se trata de una orden de cateo, una toma de testimonio, una autopsia, etc.
b) String participantes[]: Guarda los nombres de todos los implicados en el respectivo procedimiento. Es un Array para poder almacenar todos los participantes.
c) String roles[]: Guarda los roles que cumplen los participantes del procedimiento actual, o sea si son policías, testigos, el abogado,etc. Es un Array para poder almacenar todos los roles.
d) String Resultado: Guarda una breve descripción de cómo resultó el respectivo procedimiento.

2. Causa: Esta clase guardará la información de las diferentes causas que se ingresen en el programa. Sus atributos son:

a) String codigo: Guarda el código numérico que se suele usar para identificar cada caso dentro de la fiscalía. Es tipo string porque a pesar de ser un número no necesitaremos hacer ninguna operación con él
b) Fiscal encargado: Guarda los datos del fiscal que está a cargo de la causa. Es un objeto de la clase Fiscal porque será útil para las funcionalidades a implementar.

c) Procedimientos peritajes[]: Guarda en orden los procedimientos que se han realizado en su respectiva causa. Es un Array para poder almacenar todos los peritajes.
d) String estado: Guarda el estado actual de la causa.
e) String tipoCaso: Guarda la temática del delito investigado en la causa. Se incluye este dato debido a que normalmente se elige al fiscal considerando la temática que sea su especialidad (crímenes sexuales, económicos, crimen organizado, etc).
f) int distrito: Guarda el distrito en donde se desarrolla la causa. Se incluye este dato debido a que también se elige a los fiscales por la cercanía al lugar del suceso.
g) boolean asignada: Booleano que indica si la causa ya ha sido asignada a algún fiscal.

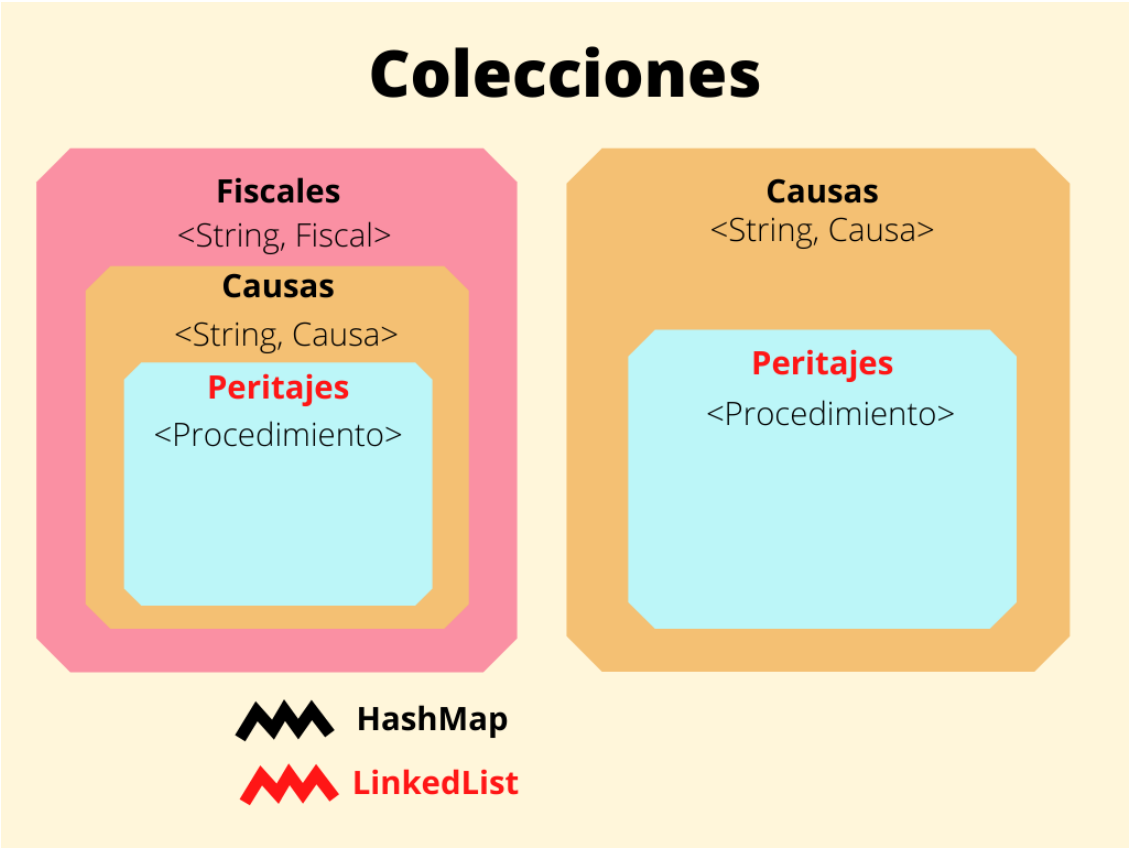
3. Fiscal: Esta clase almacena los datos de cada fiscal junto con sus causas en proceso y su especialidad, la cual será utilizada para la asignación de casos posteriormente. Sus atributos son:

a) String nombre: Almacena el nombre de cada fiscal.
b) String rut: Almacena el rut de cada fiscal para facilitar la búsqueda de estos.
c) Causa causasActuales[]: Almacena en un arreglo todas las causas que están siendo procesadas por cada fiscal.
d) String especialidad: Almacena el área de especialidad de cada fiscal, la cual será utilizada como discriminador para la asignación de los casos.
e) String región: Almacena la región donde ejerce su trabajo cada fiscal, la cual servirá posteriormente como filtro para la asignación de casos.

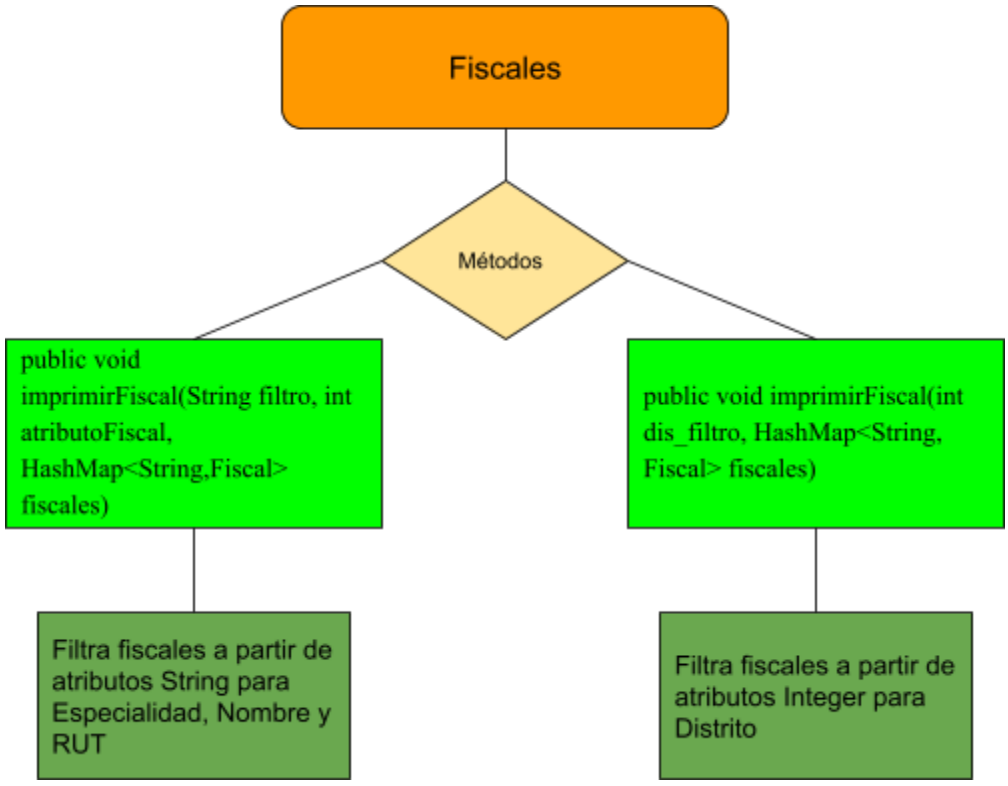
4.Fiscalía: Esta clase almacena las colecciones principales de fiscales y causas para mantener un registro dentro del programa. También almacena una contraseña que más adelante se usará para iniciar sesión como administrador dentro del programa. Sus atributos son:

a) HashMap<String,Fiscal> Fiscales: Almacena la información de todos los fiscales.
b) HashMap<String, Causa> Causas: Almacena la información de todas las causas.
c) String contraseña: Guarda la contraseña de administrador.

EP2.1 Diseño conceptual y codificación de 2 (dos) niveles de anidación de colecciones de objetos.



EP2.2 Diseño conceptual y codificación de 2 (dos) clases que utilicen sobrecarga de métodos.

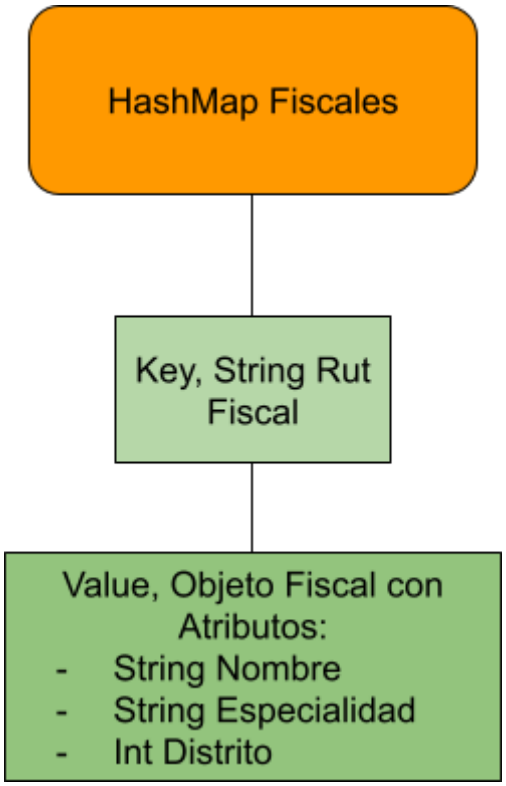


En este proyecto se hace uso de la sobreescritura de métodos varias veces, sirviendo de demostración el diagrama UML de este mismo, así que en esta sección se mostrará solamente un par de ejemplos de cómo hemos implementado la sobreescritura en las diferentes clases del programa.

La clase Fiscalía es la que más utiliza sobrecarga, pero el mejor ejemplo de esto sería el método eliminarFiscal. Cuando no se le entregan parámetros, procede a eliminar los datos del fiscal mientras llama al otro método del mismo nombre. Mientras tanto el eliminarFiscal que recibe un mapa de Causas se dedica dejar vacío el campo de encargado en cada causa que haya estado anteriormente a cargo de este fiscal, para después eliminar el mapa de causasActuales del fiscal.

En la clase Fiscal también se muestra sobrecarga en el método imprimirFiscal. Cuando este método recibe como parámetros (String filtro, int atributoFiscal, HashMap<String,Fiscal> fiscales) filtra por parámetros String e muestra por pantalla a los fiscales filtrados. En cambio, la versión que recibe de parámetros (int dis_filtro, HashMap<String,Fiscal> fiscales) se encarga de hacer esto pero filtrando por distrito,que es un parámetro numérico y no tipo String.

EP2.3 Diseño conceptual y codificación de al menos 1 clase mapa del JCF

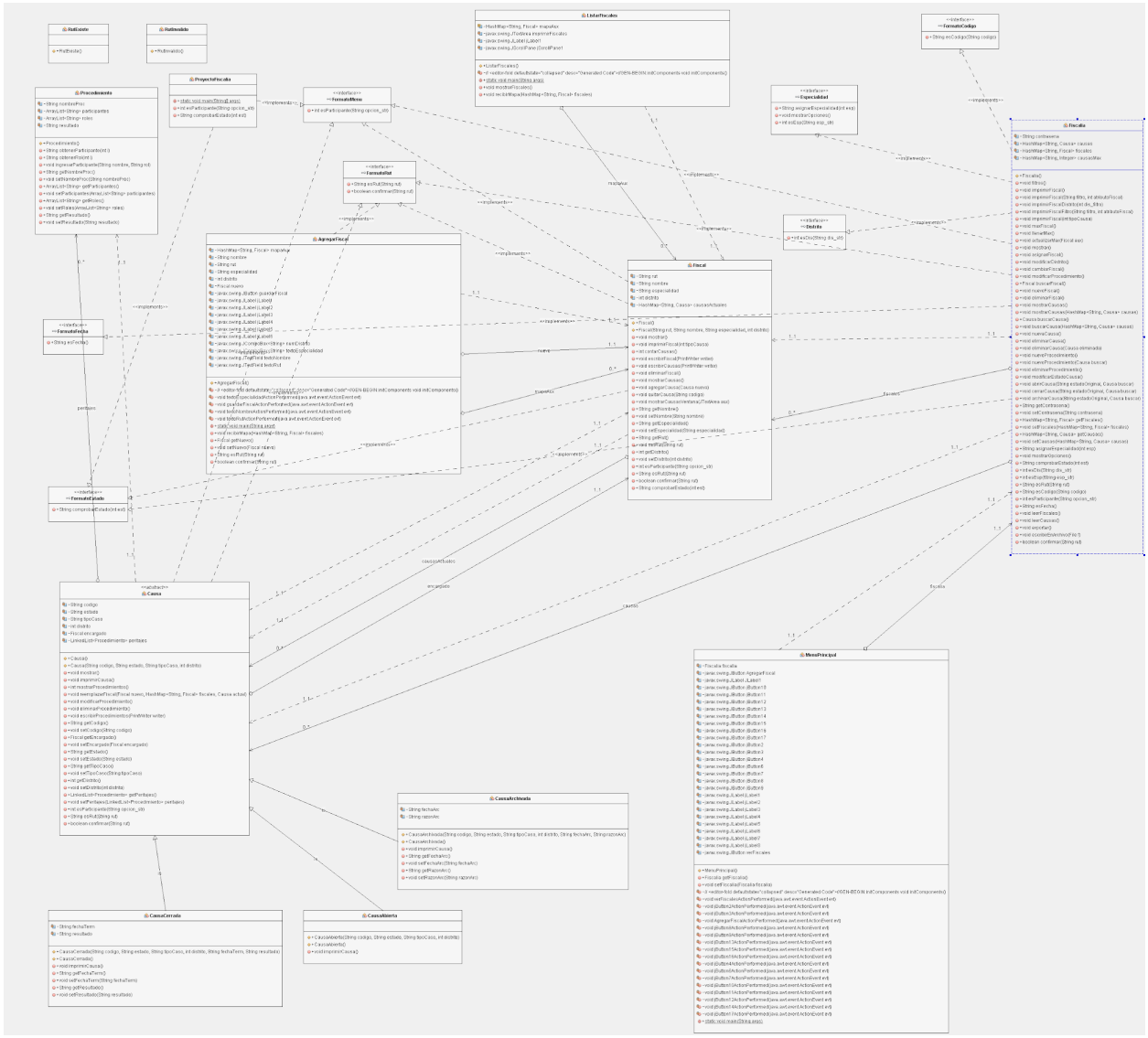


Como se muestra en la figura anterior, nuestro HashMap Fiscales, trabaja con un objeto value Fiscal con sus atributos propios y es identificado a partir de su RUT.

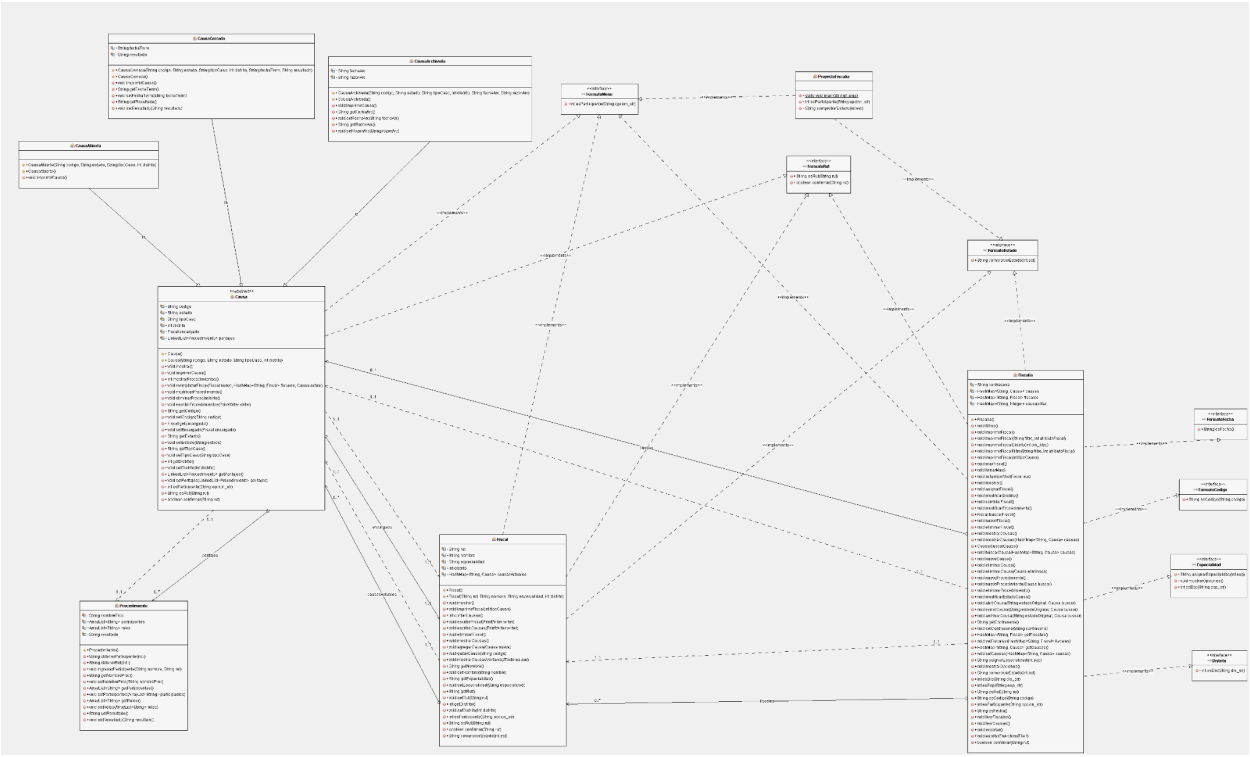
Otro ejemplo es nuestro HashMap Causa, trabaja con un objeto value Causa con sus atributos propios (ie , Estado, Tipo de Caso….) y es identificado a partir de su Código de Causa.

EP3.1.Diseño de diagrama de clases UML

Diagrama Completo



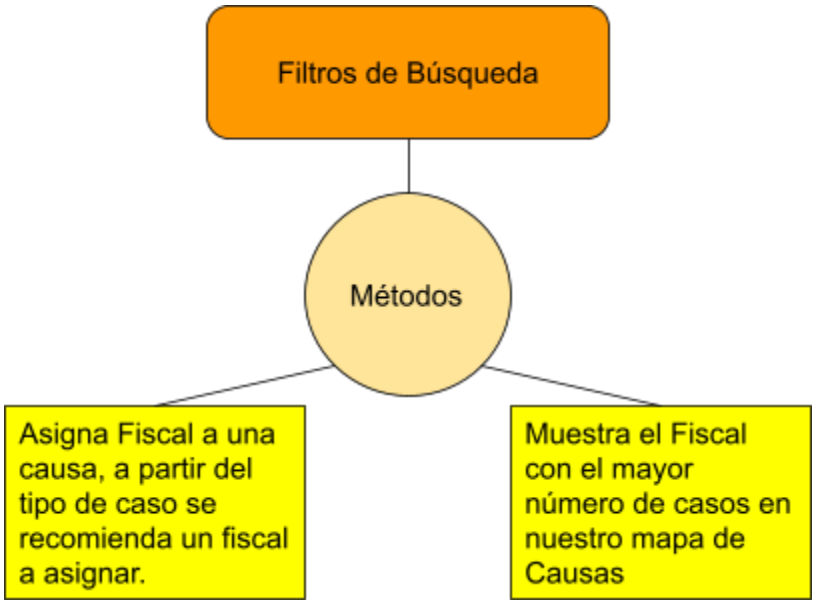
Clases de dominio y su relación con cada interfaz



Para visualizar los diagramas UML en mejor resolución, dentro de la siguiente carpeta:

...\Fiscalia\reporte\diagramas

EP4.1 Se deben incluir al menos 2 funcionalidades propias que sean de utilidad para el negocio (distintas de la inserción, edición, eliminación y reportes).



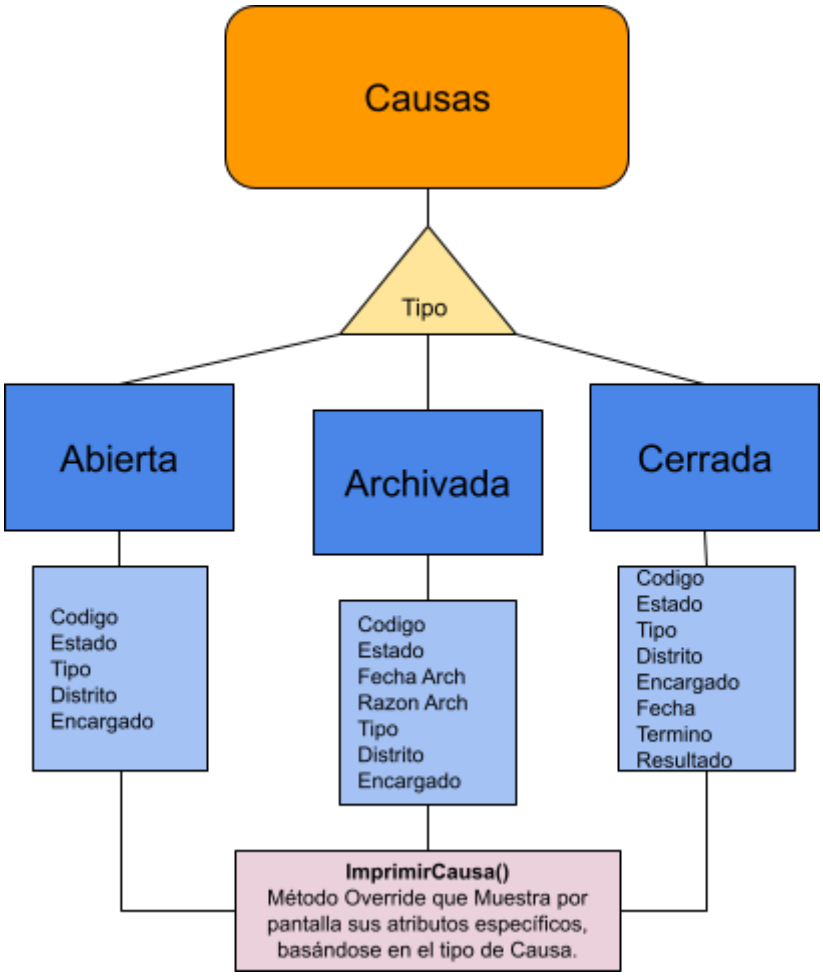
Como se puede apreciar en nuestro mapa conceptual y nuestro código, ambos métodos trabajan con las colecciones “Causa” y “Fiscal”.

El método de causa.java “public void asignarFiscal”, recorre el mapa de fiscales, utilizando la causa asignada por el usuario, a partir del tipo de caso de la causa se recomiendan algunos fiscales.

El método fiscal.java “public void maxFiscal” se encarga de buscar el Fiscal con el mayor número de casos, en estos momentos selecciona el primer Fiscal encontrado en caso de repetirse el máximo.

A su vez se encuentran métodos para filtrar un fiscal o fiscales a partir de un atributo dado (ie. Especialidad, Nombre, Distrito)

EP4.2 Diseño y codificación de 2 (dos) clases que utilicen sobreescritura de métodos.

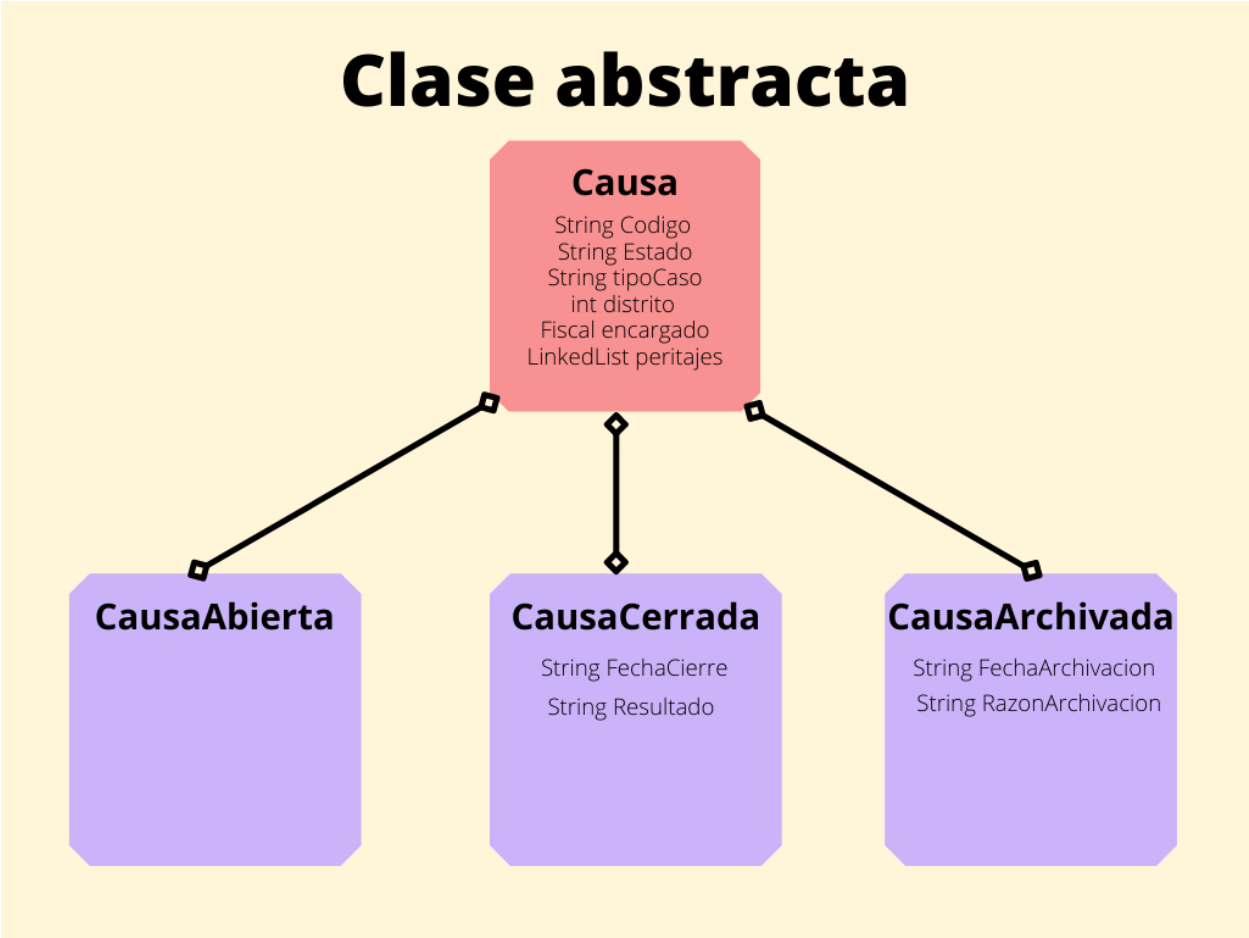


Como el ejemplo anterior, en la clase Causa el método abstracto imprimirCausa es sobreescrito por CausaAbierta, CausaCerrada y CausaArchivada dependiendo del tipo de Causa que se esté trabajando, cada método con sus atributos propios.

Adicionalmente en el uso de interfaces de verificación como “esRut”, “esDistrito” y otras son usados en la mayor parte del programa con métodos override.

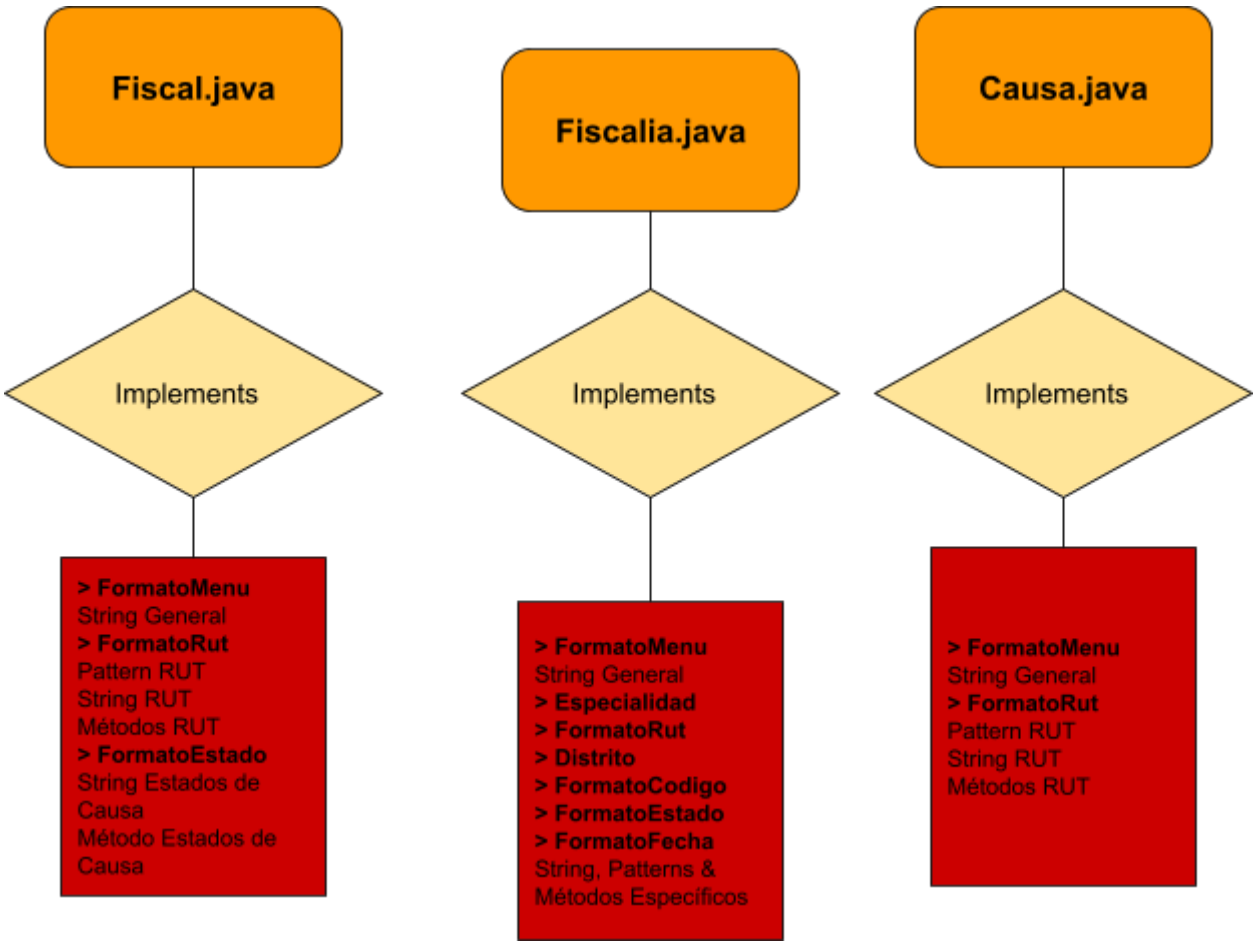
En Fiscal, se hacen uso de mismos métodos como es el caso de imprimirFiscal() con diferentes constructores y mostrar(), pero, al no ser extendida, dichos métodos no hacen uso de “override”, la razon es que Fiscal.java no necesita de los mapas de Fiscales y Causas.

EP4.3 Diseño y codificación de 1 (una) clase abstracta que sea padre de al menos 2 (dos) clases. La clase abstracta debe ser utilizada por alguna otra clase (contexto)



En el diagrama mostrado anteriormente el color salmón representa clase abstracta y el color lila las clases normales.

EP4.4 Diseño y codificación de 1 (una) interfaz que sea implementada por al menos 2 (dos) clases. La interfaz debe ser utilizada por alguna otra clase (contexto)



Dentro del proyecto se han implementado en total 7 interfaces, las cuales se explicarán a continuación y se explicará qué clases hacen uso de ellas:

-Distrito: Contiene el rango dentro del que debe estar el distrito y un método para comprobar si el distrito es válido. La implementa Fiscalía.

-Especialidad: Contiene las opciones posibles para la especialidad del Fiscal y el tipoCaso de Causa, junto con métodos para mostrar las opciones disponibles, asignar la especialidad según la opción elegida por el usuario y comprobar que una especialidad sea válida. Es implementada por Fiscalía.

-FormatoCodigo: Como dice su nombre indica el formato que debería tener el codigo de Causa junto con un método para comprobar que codigo cumpla con este formato. Es implementada por Fiscalía.

-FormatoEstado: Indica los estados posibles de una Causa y posee un método para comprobar que el estado sea válido.Es implementada por ProyectoFiscalía y Fiscalía.

-FormatoFecha: Indica el formato que debe tener una fecha para ser ingresada correctamente al sistema junto con un método que comprueba esto. Es implementada por Fiscalía.

-FormatoMenu: Contiene todas las líneas de código que se repetían constantemente en el menú de la consola y los datos para la lectura y escritura de archivos. Es implementada por Causa,Fiscal,ProyectoFiscalía y Fiscalía.

-FormatoRut: Contiene el formato que debe tener el rut y 2 métodos para comprobar que se haya ingresado un valor válido, uno para el menú en consola y otro que se usa en las ventanas. Esta interfaz es implementada por la ventana AgregarFiscal y las clases Causa,Fiscal y Fiscalia

EPB.2 Se deben implementar al menos 3 ventanas gráficas(GUIs en AWT o SWING):1 ventana de menú, 1 ventana de agregar elemento y 1 ventana de listar elementos.

En el proyecto se implementaron 5 ventanas en total, pero las solicitadas para esta sección fueron MenuPrincipal para la ventana de menú, AgregarFiscal para agregar elemento y ListarFiscales para listar elementos. Todos estos archivos pueden encontrarse en el package ventanas dentro del mismo proyecto.

EPB.3 Se debe aplicar encapsulamiento y principios OO

Para la correcta implementación del encapsulamiento y los principios de OO, eliminamos los getter y setters de mapas, nos aseguramos de que todos los atributos de las clases estuvieran en privacidad private y organizamos el código de tal forma que cada clase tenga sus propios métodos de manera que resguarde la integridad de los datos dentro del programa.

EF.2 Implementar el manejo de excepciones capturando errores potenciales específicos mediante Try-catch

Sobre todo para el control del correcto “input” del RUT, Especialidad entre otros, hemos implementado un manejo de excepciones, un ejemplo a continuación:

```
public String esRut(String rut) {
    boolean esRut;

    do {
        esRut = true;
        try {
            Matcher mat = PATRON_RUT.matcher(rut);
            if (!mat.matches()) {
                esRut = false;
                System.out.println(INCORRECTO);
                rut = LEER.nextLine();
            }
        } catch (Exception e) {
            esRut = false;
            System.out.println(INCORRECTO);
            rut = LEER.nextLine();
        }
    } while (!esRut);

    /*Retorna rut en formato correcto*/
    return rut;
}
```

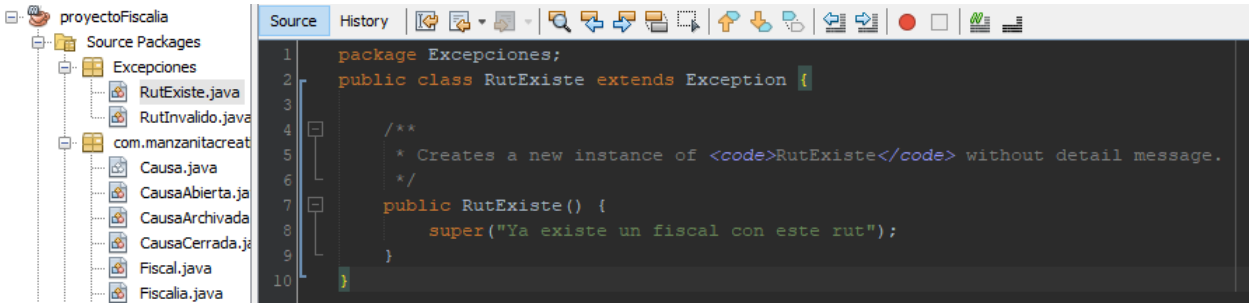
En este caso “esRut”, es un verificador universal del correcto uso “input” de usuario, utilizando try-catch en caso de un uso incorrecto, que es repetido hasta llegar al valor pedido.

EF.3 Crear 2 clases que extiendan de una excepción y que se utilicen en el programa.

Para este punto en específico hemos hecho uso de una package nueva “Excepciones” que contiene clases que extienden a Exception para hacer uso en nuestra ventana de interfaz de usuario al ingresar un código incorrecto, en caso de este ultimo ser invalido o en caso de existencia, tal como podemos apreciar en estas imágenes:

```
private void guardarFiscalActionPerformed(java.awt.event.ActionEvent evt) {
//GEN-FIRST:event_guardarFiscalActionPerformed

    nombre=textoNombre.getText();
    rut=textoRut.getText();
    especialidad=(String)textoEspecialidad.getSelectedItem();
    distrito=Integer.parseInt((String)numDistrito.getSelectedItem());
    boolean valido=confirmar(rut);
    try{
        if(!valido) throw new RutInvalido();
        nuevo= new Fiscal(nombre,rut,especialidad,distrito);
        try{
            if(mapaAux.containsKey(rut)) throw new RutExiste();
            mapaAux.put(rut, nuevo);
            System.out.println("El fiscal ha sido ingresado con exito");
        }catch (RutExiste e){
            JOptionPane.showMessageDialog(null,e.getMessage());
        }
    }catch (RutInvalido e){
        JOptionPane.showMessageDialog(null,e.getMessage());
    }
}//GEN-LAST:event_guardarFiscalActionPerformed
```



EF.4. Aplicación del patrón de diseño Strategy (Estrategia)

Para nuestro patrón strategy, como ejemplo, utilizaremos la clase abstracta “Causa.java” como clase padre, “Fiscalia.java” como base de operaciones y “CausaAbierta”, “CausaArchivada” y “CausaCerrada” como clases anidadas al método “imprimirCausa”, La cual se trata de una solución diferente dependiente del tipo de Causa que estemos trabajando.

Otro ejemplo, es la implementación de interfaces para el correcto control de nuestro programa, dentro del package “interfaces”, se encuentran diferentes soluciones para problemas de formato y verificación en las diferentes clases de dominio de nuestro programa.