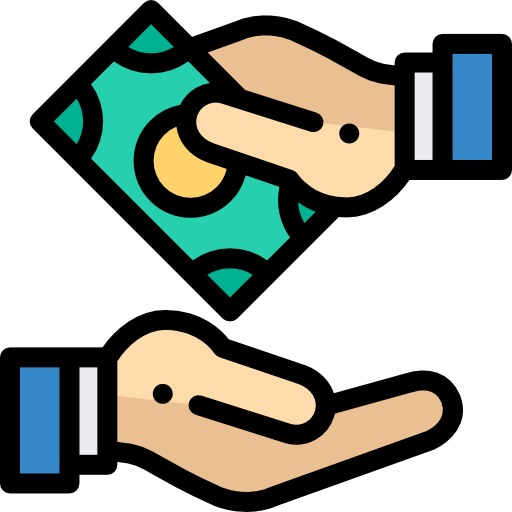


**SISTEMA DE CÁLCULO TRIBUTO**



# **INDICE**

Contenido

[**INDICE** 2](#_Toc44954422)

[**SUMARIO** 3](#_Toc44954423)

[Introducción 3](#_Toc44954424)

[Presentación 3](#_Toc44954425)

[**CAPÍTULO I** 3](#_Toc44954426)

[1. Proyecto De Investigación 3](#_Toc44954427)

[1.1. Título Del Proyecto 3](#_Toc44954428)

[1.2. Alcance 3](#_Toc44954429)

[1.3. Marco Teórico 3](#_Toc44954430)

[1.4. Objetivos 4](#_Toc44954431)

[**CAPÍTULO II** 4](#_Toc44954432)

[2. Desarrollo Funcional Del Proyecto 4](#_Toc44954433)

[2.1. Metodología de desarrollo de software del proyecto 4](#_Toc44954434)

[2.2. Requerimientos del Proyecto 5](#_Toc44954435)

[2.3. Diagrama de clases del proyecto 12](#_Toc44954436)

[2.4. Resumen De Colecciones En Java Y Forma De Aplicación 12](#_Toc44954437)

[2.5. Resumen Sobre Archivos En Java Y Forma De Aplicación 14](#_Toc44954438)

[2.6. Manual de clases 16](#_Toc44954439)

[2.7. Diagrama de clases actualizado 29](#_Toc44954440)

[2.8. Librerías Gráficas 29](#_Toc44954441)

[2.9. Pruebas de software 32](#_Toc44954442)

[**CAPÍTULO III** 33](#_Toc44954443)

[3. Conclusiones y recomendaciones 33](#_Toc44954444)

[3.1. Conclusiones 33](#_Toc44954445)

[3.2. Recomendaciones 34](#_Toc44954446)

[**CAPÍTULO IV** 34](#_Toc44954447)

[4. Bibliografía y anexos 34](#_Toc44954448)

# **SUMARIO**

## Introducción

Todos conocemos la existencia de los impuestos, hay muchos tipos, pero en este caso nos enfocaremos en el impuesto a la renta. Este impuesto se asigna en base al tipo de renta, dividiéndose consecuentemente en 5 categorías, las cuales a su vez tienen diferentes limitaciones o condiciones internas, para poder determinar cuánto es el impuesto final.

## Presentación

El cálculo del impuesto a la renta puede ser muy tedioso e inclusive complicado. Por ello, se planea realizar un programa con una interfaz sencilla de maniobrar por el usuario, que tome en cuenta todos los factores posibles como categoría, ingresos o ganancias, entre otros, para poder brindar una respuesta acertada y rápida.

# **CAPÍTULO I**

## Proyecto De Investigación

### Título Del Proyecto

Sistema de Cálculo de Tributos

### Alcance

Para lograr este proyecto necesitamos implementar un programa eficaz en Programación Orientada a Objetos con el lenguaje de programación Java, tomando en cuenta el alcance obtendremos un buen objetivo y/o resultado.

El programa estará constituido de dos partes fundamentales, la del registro de clientes nuevos, en donde se le asignará un número de RUC, tipo de categoría, datos personales. y la parte de la Emisión de Recibos por Honorarios; la cual para acceder a esta última será necesario un “Login” con datos anteriormente designados y actualizados. Todo esto, será constituido de una manera gráfica para el uso favorable del sistema.

Se invertirá esfuerzo y mucho tiempo para lograr un buen desenlace del programa requerido.

### Marco Teórico

* Combobox: Es lista desplegable, en el cual se puede ingresar “items” al inicio del programa, con la información que necesitemos. Además, le da más facilidad al usuario, ya que este solo necesita hacer clic en el combobox y posteriormente elegir la opción que requiera.
* Jlist: Es una lista, la cual podemos inicializar como una lista vacía, para luego añadir elementos en forma de cadena de caracteres, los cuales muestran los datos que hemos almacenado hasta el momento.
* Label: Son etiquetas con texto no editable por usuario, las cuales se pueden colocar cerca a otras unidades de control para que el usuario pueda saber con claridad que hace cada una de ellas.
* Button: Como su nombre indica es un botón, el cual realiza acciones determinadas en el código, cuando se le hace clic. Por ejemplo, podría haber un botón que obtenga la información anteriormente registrada y la muestre en un Jlist, para que el usuario pueda ver fácilmente los datos que se han ingresado.

### Objetivos

#### General

Programar un sistema capaz de calcular el impuesto a la renta considerando sus categorías y estándares generales, logrando implementar un proyecto eficaz y confiable.

#### Específicos

* Investigar cómo se calcula el pago del impuesto a la renta
* Diseñar una interfaz gráfica práctica para el programa en cuestión
* Definir las clases que se utilizaran en el programa, así como sus respectivos atributos y métodos.
* Aplicar los conocimientos obtenidos en el curso de Técnicas de la programación orientada a objetos en soluciones eficaces ante diversas problemáticas

# **CAPÍTULO II**

## Desarrollo Funcional Del Proyecto

### Metodología de desarrollo de software del proyecto

La metodología Scrum es una forma de trabajo en equipo que se centra en la división del trabajo en partes pequeñas llamadas sprint. Es una metodología que se utiliza en proyectos que son propensos a cambios repentinos o nuevos requerimientos solicitados por el cliente, el cual puede participar en prácticamente cualquier etapa del desarrollo. Así mismo, se le considera de desarrollo incremental, ya que va desarrollando el proyecto en pequeños bloques, revisando que funcionen, planificando cómo mejorarlos, y como se desarrollara el siguiente sprint.

### Requerimientos del Proyecto

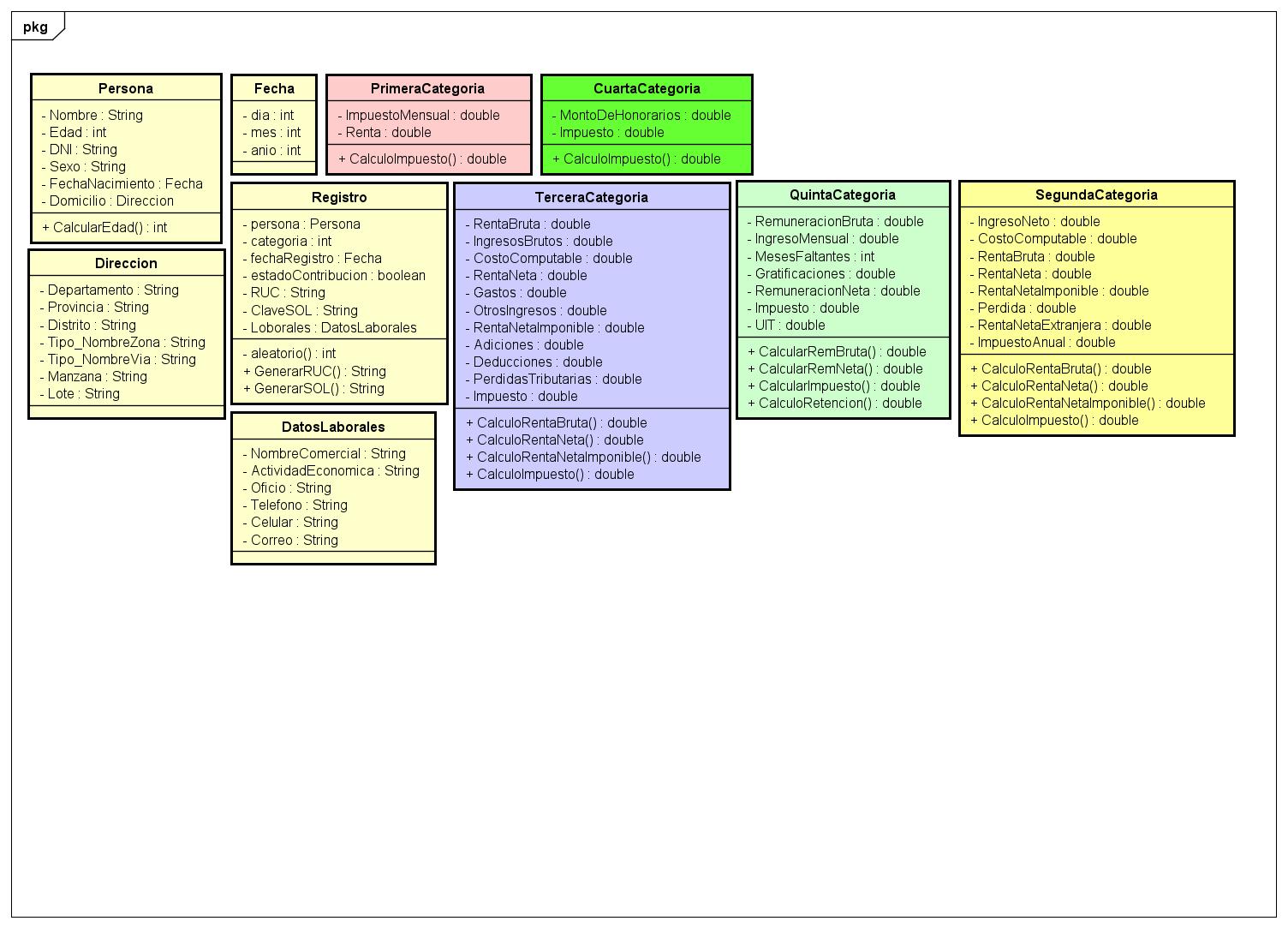
#### Funcionales

* Para el **menú** **principal**, Utilizamos un JMenuBar donde se muestre a través de un Jmenu la opción de NUEVO donde se registre el cliente, TRAMITE donde se ingrese el pago de impuesto, SCT donde se ingresara los datos del LOGIN.
* Utilizamos un JDesktopPane para crear una ventana interna de diseño metida dentro del panel.
* **Para poder registrar** a un nuevo cliente, a través de 5 radio buttons se le asigna una categoría al cliente nuevo.
* Utilizar un ScrollPane para mostrar los componentes de recepción de datos.
* Utilizar un JTextField para que el usuario pueda escribir su nombre completo y DNI .Así mismo, se validará que no ingresen caracteres erróneos respectivamente según el tipo.
* Utilizar dos radiobuttons para que el usuario pueda seleccionar el sexo correspondiente.
* Utilizar un JDateChooser para que el usuario pueda seleccionar su fecha de nacimiento correspondiente, limitando entre los años 1959 y 2001, ya legalmente se consideraría “Mayor de edad”.
* Utilizar tres JComboBox, con datos predeterminados para mostrar el Departamentos, Provincia y Distrito del usuario, respectivamente.
* Utilizar JtextField para que el usuario pueda ingresar el nombre de la zona, nombre de la vía, número de casa, Manzana y Lote en donde vive actualmente. Asimismo se valida que los datos ingresados sean los correctos, para asegurar que correspondan a un tipo de dato específico.
* Utilizar JtextField para que el usuario pueda ingresar el nombre de su empresa , su actividad económica, su oficio , su teléfono , su celular y su correo.Asimismo se valida que los datos ingresados sean los correctos, para asegurar que correspondan a un tipo de dato específico
* Utilizar un JDateChooser que mostrará los datos predeterminados según la fecha de registro en el programa
* Utilizar un JCombobox para que muestre la categoría asignada del cliente.
* Utilizar un button para guardar los datos ingresados por el usuario.
* Utilizar un Jlabel para mostrar el DNI, RUC y Clave SOL de la persona registrada.
* Utilizar un JTable para mostrar los **movimientos** del usuario de **primera** categoría, mostrando su DNI, NOMBRE, INGRESO , IMPUESTO Y FECHA DE EMISIÓN.
* Utilizar un JTable para mostrar los **movimientos** del usuario de **segunda** categoría, mostrando su DNI, NOMBRE, INGRESO , IMPUESTO Y FECHA DE EMISIÓN.
* Utilizar un JTable para mostrar los **movimientos** del usuario de **tercera** categoría, mostrando su DNI, NOMBRE, INGRESO , IMPUESTO Y FECHA DE EMISIÓN.
* Utilizar un JTable para mostrar al usuario todos sus **movimeintos** de **cuarta** categoría realizados con sus respectivos datos.
* Utilizar un JTable para mostrar los **movimientos** del usuario de **quinta** categoría, mostrando su DNI, NOMBRE, INGRESO , IMPUESTO Y FECHA DE EMISIÓN.
* El sistema debe de generar un reporte estadístico que indique la cantidad de personas correspondientes a una de las categorías de impuesto a la renta.
* Utilizar un JTabbedPane para **mostrar** **información** de como calcular el impuesto en todas las categorías de nuestro programa. Asimismo, utilizar JLabel para mostrar los textos.
* Para el **login** de **cliente**, utilizar dos JTextField para recepcionar el valor de RUC. Asimismo, validar que solo se puedan ingresar números.
* Utilizar un JTextField para recepcionar el valor de la clave. Asimismo, limitar la cantidad de caracteres a 12.
* Utilizar un JTextField para recepcionar el valor de la nueva contraseña. A su vez, limitar la cantidad de caracteres a 12.
* Utilizar un JTextField para recepcionar el valor de la contraseña a confirmar. Adicionalmente, limitar la cantidad de caracteres a 12.
* Utilizar un JButton para comparar los datos ingresados por el usuario con los anteriormente registrados, y proceder con el programa en caso de coincidencia.
* Utilizar un JButton para mostrar el panel de cambio de contraseña.
* Utilizar un JButton para guardar el valor de la nueva contraseña. Asimismo, aquí validar que la contraseña nueva y confirmar contraseña sean iguales.A su vez, validar que el ruc ingresado se encuentre registrado.
* Utilizar un JTree para permitir al usuario acceder a otros formularios fácilmente.
* Para la **actualización** de **datos**, utilizar un JTextField que permita al usuario ingresar sus nombres completos.
* Utilizar un JTextField que permita al usuario ingresar su DNI.
* Utilizar dos radiobuttons para que el usuario pueda seleccionar el sexo correspondiente.
* Utilizar un JDateChooser para que el usuario pueda seleccionar su fecha de nacimiento correspondiente, limitando entre los años 1959 y 2001, ya legalmente se consideraría “Mayor de edad”.
* Utilizar un JButton que permita al usuario ingresar sus datos de índole personal.
* Utilizar un JButton que almacene los nuevos datos de índole personal ingresados por el usuario.
* Utilizar tres JComboBox, con datos predeterminados para mostrar el Departamentos, Provincia y Distrito del usuario, respectivamente.
* Utilizar cinco JTextField para que el usuario pueda ingresar el nombre de la zona, nombre de la vía, número de casa, Manzana y Lote donde reside respectivamente. Asimismo, validar que los datos ingresados sean los correspondientes al tipo de dato que el programa espera recibir.
* Utilizar un JButton que permita al usuario ingresar los datos de su domicilio
* Utilizar un JButton que almacene los nuevos datos de domicilio ingresados por el usuario.
* Utilizar seis JTextField para que el usuario pueda ingresar el nombre comercial, actividad económica, oficio, teléfono, celular y correo, respectivamente. Asimismo, validar que los datos ingresados sean los correspondientes al tipo de dato que el programa espera recibir.
* Utilizar un JButton que permita al usuario ingresar sus datos laborales.
* Utilizar un JButton que almacene los nuevos datos laborales ingresados por el usuario.
* Para la **primera** **categoría**, utilizar un textbox para que el usuario pueda digitar la cantidad ganada por arrendamiento mensualmente, y realizar validaciones para evitar el ingreso de caracteres o valores negativos.
* Utilizar un JLabel para mostrar el impuesto calculado a partir de la cantidad introducida por el usuario en cantidad ganada por arrendamiento.
* Utilizar 2 JLabel para mostrar el nombre de la persona que está realizando su cálculo de impuesto de primera categoría.
* Utilizar 2 JLabel para mostrar la edad de la persona.
* Utilizar 2 JLabel para mostrar el dni de la persona.
* Utilizar 2 JDateChooser para mostrar al usuario la fecha actual.
* Utilizar un JLabel para mostrar al usuario la cantidad ingresada anteriormente en el textbox correspondiente a la cantidad ganada por arrendamiento.
* Utilizar un JLabel para mostrar el valor del impuesto calculado anteriormente.
* Utilizar un JButton para calcular el impuesto.
* Utilizar un JButton para guardar los datos en un archivo de texto.
* Utilizar un JButton para cerrar el formulario interno.
* Para **segunda** **categoría**, Utilizamos un textbox para ingresar el monto de Ingreso Neto. Asimismo se valida que los datos ingresados sean los correctos, para asegurar que correspondan a un tipo de dato específico.
* Utilizamos un textbox para ingresar el Costo Computable. Asimismo se valida que los datos ingresados sean los correctos, para asegurar que correspondan a un tipo de dato específico.
* Utilizamos un textbox para ingresar la Pérdida de Capital. Asimismo se valida que los datos ingresados sean los correctos, para asegurar que correspondan a un tipo de dato específico.
* Utilizamos un textbox para ingresar la Renta Neta de fuente extranjera.
* Utilizamos un textbox para ingresar el nombre del usuario. Asimismo se valida que los datos ingresados sean los correctos, para asegurar que correspondan a un tipo de dato específico.
* Utilizamos un textbox para ingresar el edad del usuario. Asimismo se valida que los datos ingresados sean los correctos, para asegurar que correspondan a un tipo de dato específico.
* Utilizamos un textbox para ingresar el dni del usuario. Asimismo se valida que los datos ingresados sean los correctos, para asegurar que correspondan a un tipo de dato específico.
* Utilizamos un label para recibir la Renta Bruta.
* Utilizamos un label para recibir la Renta Neta que se obtiene mediante la renta bruta - deducciones.
* Utilizamos un label para recibir la Renta Neta Imponible que se obtiene de la renta neta - pérdida de capital + renta neta de fuente extranjera.
* Utilizamos un label para recibir el Impuesto que viene a ser el 6.25% de la renta neta imponible.
* Utilizamos un label llamado impuesto donde se calculara el impuesto.
* Para **tercera** **categoría**, Utilizamos un textbox para ingresar el monto de Ingreso Neto. Asimismo se valida que los datos ingresados sean los correctos, para asegurar que correspondan a un tipo de dato específico.
* Utilizamos un textbox para ingresar el Costo Computable. Asimismo se valida que los datos ingresados sean los correctos, para asegurar que correspondan a un tipo de dato específico.
* Utilizamos un textbox para ingresar la Pérdida de Capital. Asimismo se valida que los datos ingresados sean los correctos, para asegurar que correspondan a un tipo de dato específico.
* Utilizamos un textbox para ingresar la Renta Neta de fuente extranjera.
* Utilizamos un textbox para ingresar el nombre del usuario. Asimismo se valida que los datos ingresados sean los correctos, para asegurar que correspondan a un tipo de dato específico.
* Utilizamos un textbox para ingresar el edad del usuario. Asimismo se valida que los datos ingresados sean los correctos, para asegurar que correspondan a un tipo de dato específico.
* Utilizamos un textbox para ingresar el dni del usuario. Asimismo se valida que los datos ingresados sean los correctos, para asegurar que correspondan a un tipo de dato específico.
* Utilizamos un label para recibir la Renta Bruta.
* Utilizamos un label para recibir la Renta Neta que se obtiene mediante la renta bruta - deducciones.
* Utilizamos un label para recibir la Renta Neta Imponible que se obtiene de la renta neta - pérdida de capital + renta neta de fuente extranjera.
* Utilizamos un label para recibir el Impuesto que viene a ser el 6.25% de la renta neta imponible.
* Utilizamos un label llamado impuesto donde se calculara el impuesto.
* Para **cuarta** **categoría**. Utilizar un JCombobox para que el usuario pueda seleccionar el tipo de documento de la empresa, ya sea DNI o RUC.
* Utilizar un JTextField para que el usuario pueda ingresar el número de documento de la empresa. Asimismo se valida que los datos ingresados sean los correctos, para asegurar que correspondan a un tipo de dato específico.
* Utilizar dos radioButtons para que el usuario pueda seleccionar el tipo de Inciso al que corresponda.
* Utilizar dos radioButtons para que el usuario pueda seleccionar si desea realizar la retención de Impuesto o no.
* Utilizar un Jcombobox para que el usuario pueda seleccionar el medio de pago del Honorario
* Utilizar un Jcombobox para que el usuario pueda seleccionar el tipo de moneda del honorario por la empresa.
* Utilizar un JTextField para registrar el monto de honorarios del usuario.Asimismo se valida que los datos ingresados sean los correctos, para asegurar que correspondan a un tipo de dato específico.
* Utilizar un JLabel para mostrar el impuesto generado por dicho Monto de honorarios.
* Utilizar un Jlabel para mostrar el Monto que va a recibir el usuario después de ejecutar el cálculo de impuesto
* Utilizar un Jbutton para calcular el impuesto generado del monto ingresado.
* Utilizar un Jbutton para guardar los datos importantes del usuario y almacenarlos en el archivo de Texto
* Utilizar un JLabel para mostrar el Nombre, edad y DNI de la persona Registrada.
* Utilizar un JDateChooser para mostrar de forma predeterminada la fecha del registro del trámite
* Utilizar un Jlabel para mostrar el tipo de documento de la empresa, su número, el tipo de renta, si desea o no realizar retención, el medio de pago del honorario, el tipo de maneda del honorario, el monto registrado, el impuesto generado, la devolución al cliente, el nombre del usuario, la edad del usuario y su DNI.
* Utilizar un JDateChooser para mostrar la fecha del registro del trámite.
* Utilizar un button para cerrar el formulario interno.
* Para la **quinta** **categoría**, utilizar un textbox para que el usuario pueda digitar sus gratificaciones y remuneraciones para el cálculo de dicho impuesto. Así mismo, se validará que no ingresen caracteres que no sean números o cantidades negativas.
* Utilizar un textbox para que el usuario pueda digitar su ingreso mensual para el cálculo de dicho impuesto. Así mismo, se validará que no ingresen caracteres que no sean números o cantidades negativas.
* Utilizar un JSpinField para que el usuario ingrese el número de meses que faltan para terminar el ejercicio gravable.Asimismo, restringimos el número máximo a 12 y el mínimo a 1, para asegurarnos que el usuario ingrese la cantidad de meses correspondiente.
* Utilizar un JLabel para mostrar el impuesto calculado a partir de las cantidades ingresadas anteriormente por el usuario.
* Utilizar 2 JLabel para mostrar el nombre de la persona que está realizando su cálculo de impuesto de primera categoría.
* Utilizar 2 JLabel para mostrar la edad de la persona.
* Utilizar 2 JLabel para mostrar el dni de la persona.
* Utilizar 2 JDateChooser para mostrar al usuario la fecha actual.
* Utilizar un JLabel para mostrar la cantidad de meses faltantes ingresada anteriormente por el usuario en el JSpinField.
* Utilizar un JLabel para mostrar la cantidad de ingreso mensual ingresada anteriormente por el usuario en el textbox respectivo.
* Utilizar un JLabel para mostrar la cantidad de gratificaciones ingresada anteriormente por el usuario.
* Utilizar un JLabel para mostrar la remuneración bruta anual.
* Utilizar un JLabel para mostrar la remuneración neta anual.
* Utilizar un JLabel para mostrar el impuesto anual.
* Utilizar un JTextArea para mostrar el valor aproximado de la retención que corresponde a cada mes del año.
* Utilizar un JButton para calcular el impuesto.
* Utilizar un JButton para guardar los datos en un archivo de texto.
* Utilizar un JButton para cerrar el formulario interno.
* Utilizar un JTable para **mostrar** la **información** de todos los usuarios registrados en nuestro archivo de texto.
* Utilizar un JTable para **mostrar** el **código** **ubigeo** con respecto al departamento o provincia ingresado.
* Utilizar un JComboBox para almacenar la lista de departamentos.
* Utilizar un JComboBox para almacenar la lista de provincias según su respectivo departamento.
* Utilizar un JButton para mostrar en la tabla todos los diferentes códigos ubigeo para el departamento ingresado.
* Utilizar un JButton para mostrar en la tabla todos los diferentes códigos ubigeo para la provincia ingresada.
* Utilizar un JTable para **mostrar** la información de **todos** los usuarios registrados en nuestro archivo de texto que corresponden a la **primera** categoría.
* Utilizar un JTable para **mostrar** la información de **todos** los usuarios registrados en nuestro archivo de texto que corresponden a la **segunda** categoría.
* Utilizar un JTable para **mostrar** los datos de **todos** los clientes pertenecientes a la **tercera** categoría, mostrando su DNI, NOMBRE, INGRESO, IMPUESTO y FECHA DE EMISIÓN
* Utilizar un JTable para **mostrar** la información de **todos** los usuarios registrados en nuestro archivo de texto que corresponden a la **cuarta** categoría.
* Utilizar un JTable para **mostrar** la información de **todos** los usuarios registrados en nuestro archivo de texto que corresponden a la **quinta** categoría.
* Para el **login** del **SCT** Utilizar un JTextfield para recepcionar la clave del administrador y darle acceso a la parte Estadística
* Utilizar un button para comprobar la clave ingresada con las cadenas de caracteres almacenadas en un fichero .dat
* Utilizar un JTree para mostrar el menú de ventanas internas al que puede acceder el Administrativo

#### No Funcionales

* Programación en lenguaje Java
* Interfaz gráfica de fácil lectura
* Registros de claves de usuários
* Utilización de documentos de texto para el registro de usuarios
* Cálculo realizado a base de datos ingresados por el usuario
* 1ra Categoría Cálculo: El impuesto mensual es el 5% de la Renta
* En la 2da Categoria la renta bruta es igual a Ingreso neto - costo computable
* Renta neta es igual a renta bruta - deducciones (20% renta bruta)
* Renta neta imponible es igual a renta neta - pérdida de capital + renta neta de fuente extranjera
* Impuesto anual es igual 6.25% de la renta neta imponible
* En la 3ra Categoría la renta bruta es igual a ingresos brutos - costo computable
* La renta neta es igual a renta bruta - gastos + otros ingresos.
* La renta neta imponible o pérdida, es igual a renta neta + adiciones - deducciones - pérdidas tributarias compensables.
* Impuesto a la renta es igual al 29.5% de la renta neta. Excepto si es persona jurídica, la cual es igual al 34.5%
* 4ta Categoría Cálculo: La retención mensual es el 8% del importe de tu recibo por honorarios.
* Si el importe es menor a 1500 no hay retención.
* En la 5ta Categoría se calcula la Remuneración bruta Anual,RemBA=(IngresoMensual\*MesesFaltantes)+Gratificaciones
* Si el RemBA es menor a 7 UIT (30100) entonces no habrá retención de quinta categoría.
* Caso contrario, se halla la Remuneración Neta Anual (RemNA), la cual es el resultado de restar la Remuneración bruta Anual con 7 UIT.
* El Impuesto Anual Proyectado es un porcentaje del RemNA, el cual varía dependiendo de su valor.
* Si el RemNA es hasta 5 UIT es el 8%, Si es mayor a 5 UIT y menor o igual a 20 UIT es el 14%, Si es mayor a 20 UIT y menor o igual a 35 UIT es el 17%, Si es mayor a 35 UIT y menor o igual a 45 UIT es el 20%, Si es mayor a 45 UIT es el 30%.
* Se utilizará un arreglo de tipo float para hallar la retención por mes: Enero,febrero,Marzo = ImpAP/12, Abril=(ImpAP-(E+F+Mar))/9, Mayo,Junio,Julio=(ImpAP-(E+F+Mar+Ab))/8, Agosto=(ImpAP-(Los meses anteriores))/5, Septiembre,octubre,Noviembre=(ImpAP-(meses anteriores))/4, Diciembre=ImpAP-(los meses anteriores).

### Diagrama de clases del proyecto



### Resumen De Colecciones En Java Y Forma De Aplicación

#### Concepto

Dichas colecciones no forman parte del lenguaje, sino que son clases definidas en el paquete java.util.

Existen muchos tipos de colecciones, entre los cuales se utiliza más los siguientes:

* Conjuntos: Estructuras de datos donde los elementos no tienen un orden y no se permiten duplicados,El marco de trabajo de colecciones contiene diferentes implementaciones de conjuntos incluyendo HashSet y TreeSet. El primer almacena los elementos en una tabla hash y el segundo en un árbol.
* Listas: Este tipo de colección se refiere a listas en las que los elementos de la colección tienen un orden, existe una secuencia de elementos. En ellas cada elemento estará en una determinada posición (índice) de la lista.
* Diccionarios:estructuras de datos donde cada elemento tiene asociado una clave que usaremos para recuperarlo
* Queue: Colección ordenada con extracción por el principio e inserción por el principio (LIFO – Last Input, First Output) o por el final (FIFO – First Input, First Output). Se permite elementos duplicados. No da excepciones cuando la cola está vacía/llena, hay métodos para interrogar, que devuelven ‘null’. Los métodos put()/take() se bloquean hasta que hay espacio en la cola/haya elementos.
* Corriente: Los streams permiten realizar operaciones funcionales sobre los elementos de las colecciones.

#### Aplicaciones y Usos:

Se podría usar un Arraylist que almacene datos de los usuarios que han realizado su pago de impuesto a la Renta. Para posteriormente poder mostrar sus nombres y a qué categoría pertenecen respectivamente.

Se podría usar un Conjunto para verificar que los datos como el DNI, no sean datos duplicados

Se podría usar Diccionarios para guardar los datos como el RUC y el nombre de la persona correspondiente y permitir así una búsqueda eficiente sobre la persona en cuestión.

### Resumen Sobre Archivos En Java Y Forma De Aplicación

#### Concepto

Un archivo es un fichero almacenado en algún recurso de memoria, en esta ocasión se utiliza para almacenar datos, aun cuando el programa deje de compilar. Estos ficheros, también poseen principales funciones como: insertar, listar, modificar, eliminar, buscar.

Para manipular ficheros, siempre tendremos que dar tres pasos:

* Abrir el fichero
* Guardar datos o leer datos
* Cerrar el fichero

Para poder trabajar con ficheros, es necesario usar la siguiente clase:

**CLASE FILE**: La clase que manipula los ficheros en Java se llama File

* Con esta clase se pueden hacer un gran número de operaciones sobre un fichero y sus propiedades, pero no se permite leer ni escribir.
* También permite obtener datos del fichero, como rutas, nombres, permisos e incluso si existe.
* El resto de las clases que manipulan ficheros parten de la existencia de una clase File, por lo que es la base de cualquier operación de manipulación de ficheros.

#### Aplicaciones y Usos:

**ESCRITURA DE ARCHIVOS**

-Para editar el archivo se usa el método write()

-Para avanzar a un a siguiente línea, se usa el método newLine()

String[ ] lineas={txtNombre.getText(), txtApellido.getText(),txtDNI.getText()};

fichero = new FileWriter("fichero\_leer.txt",true); //con el true permite “append”

// Escribimos linea a linea en el fichero

for (String linea : lineas) {

fichero.write(linea +"\t"); } fichero.write("\n");

**LECTURA DE ARCHIVOS**

Para leer de un fichero de texto usaremos "readLine()", que nos devuelve una cadena de texto (un "string"). Si ese string es null, quiere decir que se ha acabado el fichero y no se ha podido leer nada. Por eso, lo habitual es usar un "while" para leer todo el contenido de un fichero.

File fichero = new File("fichero\_leer.txt");

Scanner s = null;

// Leemos el contenido del fichero

txAR.append("... Leemos el contenido del fichero ...\n");

s = new Scanner(fichero);

// Leemos linea a linea el fichero

while (s.hasNextLine()) {

String linea = s.nextLine(); // Guardamos la linea en un String

txAR.append(linea+"\n"); // Imprimimos la linea }

**MODIFICAR DE ARCHIVOS**

Modificar un fichero de texto, consiste en leer un archivo y escribir su contenido en uno nuevo llamado X, excepto la línea a modificar que se reemplaza con la línea nueva.Luego se borra el fichero inicial y se renombra el nuevo fichero con el nombre del archivo inicial.

Lo primero, comprobamos si existe el fichero:

public class Main {

public static void main(String[] args) {

Archivos arch = new Archivos();

File fNuevo = new File("C:\\Users\\Carlos-PC\\IdeaProjects\\Herencia\\archivo.txt");

arch.Escribir(fNuevo,"arclos");

arch.modificar(fNuevo,"arclos","carlos"); } }

**BORRAR DATOS DE FICHERO**

Para poder eliminar los datos de un fichero, debemos de invocar el método delete() de la clase File. En caso de que se pueda realizar el borrado del fichero, dicho método devolverá true. En caso contrario devolverá false. Es por ello que deberemos de controlar su respuesta.

if (fichero.delete())

System.out.println("El fichero ha sido borrado satisfactoriamente");

else

System.out.println("El fichero no puede ser borrado");

### Manual de clases

#### CuartaCategoria

* **MontoDeHonorarios**: Es un atributo privado de tipo Double, el cual nos permite almacenar el valor numérico del monto de Honorarios que el usuario ingrese.
* **Impuesto**: Es un atributo privado de tipo Double, el cual nos permite almacenar el valor numérico del impuesto calculado.
* **getMontoDeHonorarios():** Es un método público de tipo Double, el cual devuelve el valor de nuestro atributo *MontoDeHonorarios .*
* **getImpuesto()**: Es un método público de tipo Double, el cual devuelve el valor de nuestro atributo *Impuesto .*
* **CuartaCategoria(double MontoDeHonorarios, double mes):** Es el constructor que utilizamos para la clase CuartaCategoria, este tiene como parámetros dos valores de tipo double, los cuales determinan los valores de un objeto de dicha clase.

#### Persona

* **Nombre:** Es un atributo privado de tipo String, el cual nos permite almacenar el valor una cadena de caracteres ingresado por el usuario, correspondiente al Nombre completo.
* **Edad:** Es un atributo privado de tipo Int, el cual nos permite almacenar el valor numérico de la edad del usuario
* **DNI:** Es un atributo privado de tipo String, el cual nos permite almacenar el valor una cadena de caracteres ingresado por el usuario, correspondiente a su DNI.
* **Sexo:** Es un atributo privado de tipo String, el cual nos permite almacenar el valor una cadena de caracteres ingresado por el usuario, correspondiente a su sexo
* **FechaNacimiento**: Es un atributo privado de tipo *Fecha*, el cual nos permite almacenar el valor numérico del dia, mes y año correspondiente.
* **Domicilio**: Es un atributo privado de tipo *Direccion*, el cual nos permite almacenar los valores de la clase anteriormente nombrada.
* **getNombre()**: Es un método público de tipo String, el cual devuelve el valor de nuestro atributo *Nombre.*
* **getEdad()**: Es un método público de tipo int, el cual devuelve el valor de nuestro atributo *Edad.*
* **getSexo()**: Es un método público de tipo String, el cual devuelve el valor de nuestro atributo *Sexo.*
* **getFechaNacimiento()**: Es un método público de tipo Fecha, el cual devuelve el valor de nuestro atributo *FechaNacimiento.*
* **getDomicilio()**: Es un método público de tipo Direccion, el cual devuelve el valor de nuestro atributo *Domicilio.*
* **setNombre(String Nombre):** Es un método público de tipo void, este tiene como parámetro un dato de tipo String, el cual será definido como el nuevo valor de nuestro atributo *Nombre*.
* **setEdad(int Edad):** Es un método público de tipo void, este tiene como parámetro un dato de tipo int, el cual será definido como el nuevo valor de nuestro atributo *Edad*.
* **setSexo(String Sexo):** Es un método público de tipo void, este tiene como parámetro un dato de tipo String, el cual será definido como el nuevo valor de nuestro atributo *Sexo*.
* **setFechaNacimiento(Fecha FechaNacimiento):** Es un método público de tipo void, este tiene como parámetro un dato de tipo Fecha, el cual será definido como el nuevo valor de nuestro atributo *FechaNacimiento*.
* **setDomicilio(Direccion Domicilio):** Es un método público de tipo void, este tiene como parámetro un dato de tipo Direccion, el cual será definido como el nuevo valor de nuestro atributo *Domicilio*.
* **Persona(String Nombre, int Edad, String Sexo, Fecha FechaNacimiento, Direccion Domicilio):** Es el constructor que utilizamos para la clase Persona, este tiene como parámetros dos valores de tipo String, uno de tipo int, uno de tipo Fecha y uno de tipo Domicilio, los cuales determinan los valores de un objeto de dicha clase.

#### DatosLaborales

* **NombreComercial:** Es un atributo privado de tipo String, el cual nos permite almacenar el valor una cadena de caracteres ingresado por el usuario, correspondiente al Nombre Comercial.
* **ActividadEconomica:** Es un atributo privado de tipo String, el cual nos permite almacenar el valor una cadena de caracteres ingresado por el usuario, correspondiente a la Actividad Económica.
* **Oficio:** Es un atributo privado de tipo String, el cual nos permite almacenar el valor una cadena de caracteres ingresado por el usuario, correspondiente al Oficio.
* **Telefono:** Es un atributo privado de tipo String, el cual nos permite almacenar el valor una cadena de caracteres ingresado por el usuario, correspondiente al Telefono.
* **Celular:** Es un atributo privado de tipo String, el cual nos permite almacenar el valor una cadena de caracteres ingresado por el usuario, correspondiente al número de Celular.
* **Correo:** Es un atributo privado de tipo String, el cual nos permite almacenar el valor una cadena de caracteres ingresado por el usuario, correspondiente al Correo.
* **getNombreComercial()**: Es un método público de tipo String, el cual devuelve el valor de nuestro atributo *NombreComercial.*
* **getActividadEconomica()**: Es un método público de tipo String, el cual devuelve el valor de nuestro atributo *ActividadEconomica.*
* **getOficio()**: Es un método público de tipo String, el cual devuelve el valor de nuestro atributo *Oficio.*
* **getTelefono()**: Es un método público de tipo String, el cual devuelve el valor de nuestro atributo *Telefonol.*
* **getCelular()**: Es un método público de tipo String, el cual devuelve el valor de nuestro atributo *Celular.*
* **getCorreo()**: Es un método público de tipo String, el cual devuelve el valor de nuestro atributo *Correo.*
* **setNombreComercial(String NombreComercial):** Es un método público de tipo void, este tiene como parámetro un dato de tipo String, el cual será definido como el nuevo valor de nuestro atributo *NombreComercial*.
* **setActividadEconomica(String ActividadEconomica):** Es un método público de tipo void, este tiene como parámetro un dato de tipo String, el cual será definido como el nuevo valor de nuestro atributo *ActividadEconomica*.
* **setOficio(String Oficio):** Es un método público de tipo void, este tiene como parámetro un dato de tipo String, el cual será definido como el nuevo valor de nuestro atributo *Oficio*.
* **setTelefono(String Telefono):** Es un método público de tipo void, este tiene como parámetro un dato de tipo String, el cual será definido como el nuevo valor de nuestro atributo *Telefono*.
* **setCelular(String Sexo):** Es un método público de tipo void, este tiene como parámetro un dato de tipo String, el cual será definido como el nuevo valor de nuestro atributo *Celular*.
* **setCorreo(String Correo):** Es un método público de tipo void, este tiene como parámetro un dato de tipo String, el cual será definido como el nuevo valor de nuestro atributo *Correo*.
* **DatosLaborales(String NombreComercial, String ActividadEconomica, String Oficio, String Telefono, String Celular, String Correo):** Es el constructor que utilizamos para la clase DatosLaborales, este tiene como parámetros seis valores de tipo String, los cuales determinan los valores de un objeto de dicha clase.

#### Fecha:

* **dia:** Es un atributo privado de tipo Int, el cual nos permite almacenar el valor numérico del día que el usuario ingrese .
* **mes:** Es un atributo privado de tipo Int, el cual nos permite almacenar el valor numérico del mes que el usuario ingrese.
* **anio:** Es un atributo privado de tipo Int, el cual nos permite almacenar el valor numérico del año que el usuario ingrese.
* **getdia():** Es un método público de tipo Int, el cual devuelve el valor de nuestro atributo *día*.
* **getmes():** Es un método público de tipo Int, el cual devuelve el valor de nuestro atributo *mes*.
* **getanio():** Es un método público de tipo Int, el cual devuelve el valor de nuestro atributo *anio*.
* **Fecha(int dia,int mes,int anio):** Es el constructor que utilizamos para la clase Fecha, este tiene como parámetros tres valores de tipo Int, los cuales determinan los valores de un objeto de dicha clase.

#### PrimeraCategoria

* **Renta:** Es un atributo privado de tipo Double, el cual nos permite almacenar el valor      numérico del ingreso por arrendamiento ingresado por el usuario.
* **ImpuestoMensual:** Es un atributo privado de tipo Double, el cual nos permite almacenar el valor numérico del resultado de nuestro método *CalculoImpuesto.*
* **CalculoImpuesto():** Es un método público de tipo Double, el cual devuelve el resultado de multiplicar nuestro atributo *Renta* por 0.05.
* **getRenta():** Es un método público de tipo Double, el cual devuelve el valor de nuestro atributo *Renta*.
* **getImpuestoMensual():** Es un método público de tipo Double, el cual devuelve el valor de nuestro atributo *ImpuestoMensual*.
* **PrimeraCategoria(double Renta):** Es el constructor que utilizamos para la clase PrimeraCategoria, Este tiene como parámetros un único valor de tipo Double, el cual determina el valor del atributo *Renta* en la creación de un objeto de esta clase.

#### Direccion:

* **Departamento:** Es un atributo privado de tipo String, el cual nos permite almacenar una cadena de caracteres ingresado por el usuario, correspondiente al departamento de residencia.
* **Provincia:** Es un atributo privado de tipo String, el cual nos permite almacenar una cadena de caracteres ingresado por el usuario, correspondiente a la provincia de residencia.
* **Distrito:** Es un atributo privado de tipo String, el cual nos permite almacenar una cadena de caracteres ingresado por el usuario, correspondiente al distrito de residencia.
* **Tipo\_NombreZona:** Es un atributo privado de tipo String, el cual nos permite almacenar una cadena de caracteres ingresado por el usuario, correspondiente al tipo y nombre de zona.
* **Tipo\_NombreVia:** Es un atributo privado de tipo String, el cual nos permite almacenar una cadena de caracteres ingresado por el usuario, correspondiente al tipo y nombre de via.
* **Manzana:** Es un atributo privado de tipo String, el cual nos permite almacenar una cadena de caracteres ingresado por el usuario, correspondiente a la manzana de su zona de residencia.
* **Lote:** Es un atributo privado de tipo String, el cual nos permite almacenar una cadena de caracteres ingresado por el usuario, correspondiente al lote de su zona de residencia.
* **Numero:** Es un atributo privado de tipo Int, el cual nos permite almacenar el valor numérico del número de la residencia ingresado por el usuario.
* **getDepartamento():** Es un método público de tipo String, el cual devuelve el valor de nuestro atributo *Departamento*.
* **getProvincia():** Es un método público de tipo String, el cual devuelve el valor de nuestro atributo *Provincia*.
* **getDistrito():** Es un método público de tipo String, el cual devuelve el valor de nuestro atributo *Distrito*.
* **getTipo\_NombreZona():** Es un método público de tipo String, el cual devuelve el valor de nuestro atributo *Tipo\_NombreZona*.
* **getTipo\_NombreVia():** Es un método público de tipo String, el cual devuelve el valor de nuestro atributo *Tipo\_NombreVia*.
* **getManzana():** Es un método público de tipo String, el cual devuelve el valor de nuestro atributo *Manzana*.
* **getLote():** Es un método público de tipo String, el cual devuelve el valor de nuestro atributo *Lote*.
* **getNumero():** Es un método público de tipo Int, el cual devuelve el valor de nuestro atributo *Numero*.
* **setDepartamento(String Departamento):** Es un método público de tipo void, este tiene como parámetro un dato de tipo String, el cual será definido como el nuevo valor de nuestro atributo *Departamento*.
* **setProvincia(String Provincia):** Es un método público de tipo void, este tiene como parámetro un dato de tipo String, el cual será definido como el nuevo valor de nuestro atributo *Provincia*.
* **setDistrito(String Distrito):** Es un método público de tipo void, este tiene como parámetro un dato de tipo String, el cual será definido como el nuevo valor de nuestro atributo *Distrito*.
* **setTipo\_NombreZona(String Tipo\_NombreZona):** Es un método público de tipo void, este tiene como parámetro un dato de tipo String, el cual será definido como el nuevo valor de nuestro atributo *Tipo\_NombreZona*.
* **setTipo\_NombreVia(String Tipo\_NombreVia):** Es un método público de tipo void, este tiene como parámetro un dato de tipo String, el cual será definido como el nuevo valor de nuestro atributo *Tipo\_NombreVia*.
* **setManzana(String Manzana):** Es un método público de tipo void, este tiene como parámetro un dato de tipo String, el cual será definido como el nuevo valor de nuestro atributo *Manzana*.
* **setLote(String Lote):** Es un método público de tipo void, este tiene como parámetro un dato de tipo String, el cual será definido como el nuevo valor de nuestro atributo *Lote*.
* **setNumero(int Numero):** Es un método público de tipo void, este tiene como parámetro un dato de tipo Int, el cual será definido como el nuevo valor de nuestro atributo *Numero*.
* **Direccion(String Departamento,String Provincia,String Distrito,String Tipo\_NombreZona,String Tipo\_NombreVia,String Manzana,String Lote, int Numero):** Es el constructor que utilizamos para la clase Dirección, este tiene como parámetros 6 datos de tipo String y 1 de tipo Int, los cuales determinan los valores de todos los atributos de un objeto de dicha clase.

#### QuintaCategoria:

* **RemuneracionBruta:** Es un atributo privado de tipo Double,el cual nos permite almacenar el valor numérico del resultado de nuestro método *CalcularRemBruta*.
* **IngresoMensual:** Es un atributo privado de tipo Double, el cual nos permite almacenar el valor numérico del ingreso mensual ingresado por el usuario.
* **MesesFaltantes:** Es un atributo privado de tipo Int, el cual nos permite almacenar la cantidad de meses faltantes para terminar el ejercicio gravable, la cual es ingresada por el usuario.
* **Gratificaciones:**  Es un atributo privado de tipo Double, el cual nos permite almacenar el valor numérico de las gratificaciones, el cual es ingresado por el usuario.
* **RemuneracionNeta:** Es un atributo privado de tipo Double,el cual nos permite almacenar el valor numérico del resultado de nuestro método *CalcularRemNeta*.
* **Impuesto:** Es un atributo privado de tipo Double,el cual nos permite almacenar el valor numérico del resultado de nuestro método *CalcularImpuesto*.
* **UIT:**  Es un atributo privado de tipo Double, el cual nos permite almacenar el valor numérico de las UIT, el cual es definido por el sistema en la construcción del objeto.
* **Retencion:**  Es un atributo privado de tipo Arreglo de Double, el cual nos permite almacenar el valor numérico de la retención por mes, los cuales son definidos por nuestro método *CalculoRetencion.*
* **getRemuneracionBruta():** Es un método público de tipo Double, el cual devuelve el valor de nuestro atributo *RemuneracionBruta*.
* **getIngresoMensual():** Es un método público de tipo Double, el cual devuelve el valor de nuestro atributo *IngresoMensual*.
* **getMesesFaltantes():** Es un método público de tipo Int, el cual devuelve el valor de nuestro atributo *MesesFaltantes*.
* **getGratificaciones():** Es un método público de tipo Double, el cual devuelve el valor de nuestro atributo *Gratificaciones*.
* **getRemuneracionNeta():** Es un método público de tipo Double, el cual devuelve el valor de nuestro atributo *RemuneracionNeta*.
* **getImpuesto():** Es un método público de tipo Double, el cual devuelve el valor de nuestro atributo *Impuesto*.
* **getUIT():** Es un método público de tipo Double, el cual devuelve el valor de nuestro atributo *UIT*.
* **getRetencion(int pos):** Es un método público de tipo Double, el cual devuelve el valor de uno de los elementos de nuestro atributo *Retencion,* tomando en cuenta el parametro ingresado como la posicion del objeto.
* **CalcularRemBruta():** Es un método público de tipo Double, el cual devuelve el resultado de añadir el valor de nuestro atributo *Gratificaciones* al resultado de la multiplicación de nuestros atributos *IngresoMensual* y *MesesFaltantes*.
* **CalcularRemNeta():** Es un método público de tipo Double, el cual devuelve el resultado de restar el valor de nuestro atributo *RemuneracionBruta* con 7 veces el valor de nuestro atributo *UIT*.
* **CalcularImpuesto():** Es un método público de tipo Double, el cual devuelve un porcentaje del valor de nuestro atributo *RemuneracionNeta*, dependiendo del valor de este.
* **CalculoRetencion():** Es un método público de tipo void, el cual define el valor de cada una de las posiciones de nuestro atributo *Retencion* de tipo arreglo Double, realizando cálculos utilizando el valor de nuestro atributo *Impuesto*.
* **QuintaCategoria(Double IngresoMensual,Int MesesFaltantes,Double Gratificaciones):** Es el constructor que utilizamos para la clase QuintaCategoria, este tiene como parámetros 2 datos de tipo Double y 1 de tipo Int, los cuales determinan los valores de los atributos *IngresoMensual*, *MesesFaltantes* y *Gratificaciones*. A su vez, define de manera interna el valor del atributo *UIT*.

#### SegundaCategoria

* **IngresoNeto:** es un atributo privado de tipo double que nos permitirá almacenar la cantidad total de beneficios del usuario.
* **CostoComputable:** es un atributo privado de tipo double que nos permite su almacenamiento por el usuario.
* **RentaBruta**: es un atributo privado de tipo double el cual nos permite almacenar el valor numérico del resultado de nuestro método CalculoRentaBruta().
* **RentaNeta:** es un atributo privado de tipo double el cual nos permite almacenar el valor numérico del resultado de nuestro método CalculoRentaNeta().
* **RentaNetaImponible:** es un atributo privado de tipo double el cual nos permite almacenar el valor numérico del resultado de nuestro método **CalculoRentaNetaImponible().**
* Perdida: Es un atributo privado de tipo Double, el cual nos permite almacenar el valor numérico de la pérdida ingresado por el usuario.
* **CalculoRentaBruta():** Es un método público de tipo Double, el cual nos devuelve el resultado de calculo de la renta bruta.
* **CalculoRentaNeta():** Es un método público de tipo Double, el cual nos devuelve el resultado de calculo de la renta neta.
* **CalculoRentaNetaImponible():**Es un método público de tipo Double, el cual nos devuelve el resultado de calculo de la renta neta imponible.
* **CalculoImpuesto():** Es un método público de tipo Double, el cual devuelve el resultado de multiplicar nuestro atributo Renta por 0.05.
* **getIngresoNeto:** Es un método privado de tipo Double, el cual devuelve el valor de nuestro atributo IngresoNeto.
* **getCostoComputable:** Es un método privado de tipo Double, el cual devuelve el valor de nuestro atributo CostoComputable.
* **getRentaBruta:** Es un método privado de tipo Double, el cual devuelve el valor de nuestro atributo rentaBruta.
* **getRentaNeta**: Es un método privado de tipo Double, el cual devuelve el valor de nuestro atributo RentaNeta.
* **getRentaNetaImponible:** Es un método privado de tipo Double, el cual devuelve el valor de nuestro atributo RentaNetaImponible.
* **setIngresoNeto (Double IngresoNeto):** Es un método público de tipo void, este tiene como parámetro un dato de tipo Double, el cual será definido como el nuevo valor de nuestro atributo IngresoNeto
* **setCostoComputable(Double CostoComputable):** Es un método público de tipo void, este tiene como parámetro un dato de tipo Double, el cual será definido como el nuevo valor de nuestro atributo CostoComputable.
* **setRentaBruta(Double IRentaBruta**): Es un método público de tipo void, este tiene como parámetro un dato de tipo Double, el cual será definido como el nuevo valor de nuestro atributo RentaBruta.
* **setRentaNeta(Double RentaNeta):** Es un método público de tipo void, este tiene como parámetro un dato de tipo Double, el cual será definido como el nuevo valor de nuestro atributo RentaNeta.
* **SegundaCategoria(Double IngresoNeto, Double CostoComputable, Double RentaBruta, Double RentaNeta, Double RentaNetaImponible, Double Perdida):** Es el constructor que utilizamos para la clase SegundaCategoria, este tiene como parámetros 6 datos de tipo Double, los cuales determinan los valores de todos los atributos de un objeto de dicha clase.

#### TerceraCategoria

* **RentaBruta**: Es un atributo privado de tipo Double,el cual nos permite almacenar el valor numérico del resultado de nuestro método CalcularRentaBruta.
* **IngresosBrutos**: Es un atributo privado de tipo Double.
* **RentaNeta**: Es un atributo privado de tipo Double,el cual nos permite almacenar el valor numérico del resultado de nuestro método CalculoRentaNeta.
* **Gastos**: Es un atributo privado de tipo Double, el cual nos permite almacenar el valor numérico Gastos, el cual es ingresado por el usuario.
* **OtrosIngresos**: Es un atributo privado de tipo Double, el cual nos permite almacenar el valor numérico de las OtrosGastos, el cual es ingresado por el usuario.
* **RentaNetaImponible**: Es un atributo privado de tipo Double,el cual nos permite almacenar el valor numérico del resultado de nuestro método CalculoRentaNetaImponible.
* **Adiciones**: Es un atributo privado de tipo Double, el cual nos permite almacenar el valor numérico de las Adiciones, el cual es ingresado por el usuario.
* **Deducciones**: Es un atributo privado de tipo Double, el cual nos permite almacenar el valor numérico de las Deducciones, el cual es ingresado por el usuario.
* **PerdidasTributarias**: Es un atributo privado de tipo Double, el cual nos permite almacenar el valor numérico de las PerdidasTributarias, el cual es ingresado por el usuario.
* **Impuesto**: Es un atributo privado de tipo Double,el cual nos permite almacenar el valor numérico del resultado de nuestro método CalculoImpuesto.
* **TercerCategoria(Double RentaBruta, Double IngresosBrutos, Double CostoComputable, Double RentaNeta, Double Gastos, Double OtrosIngresos, Double RentaNetaImponible, Double Adiciones, Double Deducciones, Double PerdidasTributarias, Double Impuesto**) Es el constructor que utilizamos para la clase TerceraCategoria, este tiene como parámetros 11 datos de tipo Double, los cuales determinan los valores de todos los atributos de un objeto de dicha clase.
* **getRentaBruta**(): Es un método público de tipo double, el cual devuelve el valor de nuestro atributo RentaBruta.
* **getIngresosBrutos**(): Es un método público de tipo double, el cual devuelve el valor de nuestro atributo IngresosBrutos.
* **getCostosComputables**(): Es un método público de tipo double, el cual devuelve el valor de nuestro atributo CostosComputables.
* **getRentaNeta**(): Es un método público de tipo double, el cual devuelve el valor de nuestro atributo RentaNeta.
* **getGastos**(): Es un método público de tipo double, el cual devuelve el valor de nuestro atributo Gastos
* **getOtrosIngresos**(): Es un método público de tipo double, el cual devuelve el valor de nuestro atributo OtrosImpuestos.
* **getRentaNetaImponible**():Es un método público de tipo double, el cual devuelve el valor de nuestro atributo RentaNetaImponible.
* **getAdiciones**():Es un método público de tipo double, el cual devuelve el valor de nuestro atributo Adiciones.
* **getDeducciones**():Es un método público de tipo double, el cual devuelve el valor de nuestro atributo Deducciones.
* **getPerdidaTributaria**():Es un método público de tipo double, el cual devuelve el valor de nuestro atributo PerdidaTributaria.
* **getImpuesto**():Es un método público de tipo double, el cual devuelve el valor de nuestro atributo Impuesto
* **CalculoRentaBruta**():Es un método público de tipo Double,
* **CalculoRentaNeta**():Es un método público de tipo Double,
* **CalculoRentaNetaImponible**():Es un método público de tipo Double,
* **CalculoImpuesto**():Es un método público de tipo Double,

#### Tributo

* **Contribuyente**: Es un atributo privado de tipo persona, el cual nos permite almacenar una cadena de caracteres ingresado por el usuario, correspondiente a Contribuyente.
* **FechaEmicion**: Es un atributo privado de tipo Fecha, el cual nos permite almacenar una cadena de caracteres ingresado por el usuario, correspondiente a FechaEmicion.
* **Tributo ( Contribuyente : persona, FechaEmision : Fecha)** Es el constructor que utilizamos para la clase Tributo, este tiene como parámetros 2 datos de tipo persona y Fecha, los cuales determinan los valores de todos los atributos de un objeto de dicha clase.
* **getContribuyente**():Es un método público de tipo Persona, el cual devuelve el valor de nuestro atributo Contribuyente.
* **getFechaEmision**():Es un método público de tipo Persona, el cual devuelve el valor de nuestro atributo FechaEmision.
* **getCategoria1**():Es un método público de tipo PrimeraCategoria, el cual devuelve el valor de nuestro atributo Categoria1.
* **getCategoria2**():Es un método público de tipo SegundaCategoria, el cual devuelve el valor de nuestro atributo Categoria2.
* **getCategoria3**():Es un método público de tipo terceraCategoria, el cual devuelve el valor de nuestro atributoCategoria3.
* **getCategoria4**():Es un método público de tipo CuartaCategoria, el cual devuelve el valor de nuestro atributoCategoria4.
* **getCategoria5**():Es un método público de tipo QuintaCategoria, el cual devuelve el valor de nuestro atributoCategoria5.

#### Registro

* **Categoria**: Es un atributo privado de tipo int, el cual nos permite almacenar el valor numérico del ingreso por tipo de categoria ingresado por el usuario.
* **EstadoConstitucional**: Es un atributo privado de tipo boolean, el cual nos permite almacenar el valor numérico del ingreso por tipo de EstadoConstitucional ingresado por el usuario.
* **RUC** : es un atributo privado de tipo String, el cual nos permite almacenar los datos ingresados por el usuario.
* **ClaveSOL**: es un atributo privado de tipo String, el cual nos permite almacenar los datos ingresados por el usuario.
* **Categoria ( persona : persona, Categoria : int, FechaRegistro : Fecha, Laborales : DatosLaborales)** Es el constructor que utilizamos para la clase Registro, este tiene como parámetros 4 datos de tipo Persona, int,Fecha, DatosLaborales, los cuales determinan los valores de todos los atributos de un objeto de dicha clase.
* **setPersona(persona : persona) :** Es un método público de tipo void, este tiene como parámetro un dato de tipo persona, el cual será definido como el nuevo valor de nuestro atributo Persona.
* **setCategoria(Categoria : int):** Es un método público de tipo void, este tiene como parámetro un dato de tipo Ctegoria, el cual será definido como el nuevo valor de nuestro atributo Ctegoria.
* **setEstadoContribucion(EstadoContribucion : boolean)** : Es un método público de tipo void, este tiene como parámetro un dato de tipo EstadoContribucion, el cual será definido como el nuevo valor de nuestro atributo EstadoContribucion.
* **setClaveSOL(ClaveSOL : String**) : Es un método público de tipo void, este tiene como parámetro un dato de tipo ClaveSOL, el cual será definido como el nuevo valor de nuestro atributo ClaveSOL.
* **setLaborales**(Laborales : DatosLaborales) : Es un método público de tipo void, este tiene como parámetro un dato de tipo Laborales, el cual será definido como el nuevo valor de nuestro atributo Laborales.
* **getpersona**() : Es un método público de tipo Persona, el cual devuelve el valor de nuestro atributo persona.
* **getCategoria**() : Es un método público de tipo int, el cual devuelve el valor de nuestro atributo Categoria.
* **getFechaRegistro**() : Es un método público de tipo Fecha, el cual devuelve el valor de nuestro atributo FechaRegistro.
* **getEstadoContribucion**() : Es un método público de tipo boolean, el cual devuelve el valor de nuestro atributo EstadoContribucion.
* **getRUC**() : Es un método público de tipo String, el cual devuelve el valor de nuestro atributo RUC.
* **getClaveSOL**() : Es un método público de tipo String, el cual devuelve el valor de nuestro atributo ClaveSOL.
* **getLaborales**() : Es un método público de tipo DatosLaborales, el cual devuelve el valor de nuestro atributo Laborales.
* **Aleatorio**() : Es un método público de tipo int, el cual devuelve el valor de aleatorio.
* **GenerarRUC**() : Es un método público de tipo String, el cual devuelve el valor de nuestro atributo RUC.
* **GenerarSOL()** : Es un método público de tipo String, el cual devuelve el valor de nuestro atributo SOL.

#### Admin

* **Nombre** : es un atributo privado de tipo String, el cual nos permite almacenar los datos de nombres obtenidos en el fichero Clave$.
* **Contra** : es un atributo privado de tipo String, el cual nos permite almacenar los datos de contraseñas obtenidos en el fichero Clave$.
* **getNombre**() : Es un método público de tipo String, el cual devuelve el valor de nuestro atributo Nombre.
* **getContra**() : Es un método público de tipo String, el cual devuelve el valor de nuestro atributo Contra.
* **setNombre(Nombre : String):** Es un método público de tipo void, este tiene como parámetro un dato de tipo Nombre, el cual será definido como el nuevo valor de nuestro atributo Nombre.
* **setContra(Contra : String):** Es un método público de tipo void, este tiene como parámetro un dato de tipo Contra, el cual será definido como el nuevo valor de nuestro atributo Contra.
* **Admin ( String : Nombre, String : Contra)** Es el constructor que utilizamos para la clase Admin, este tiene como parámetros 2 datos de tipo String, los cuales determinan los valores de todos los atributos de un objeto de dicha clase.

#### Arreglo1Tributo

* **aTributo:** Es un atributo privado Arraylist de objetos de tipo Tributo, el cual nos permite almacenar la información de los trámites de quinta categoría realizados.
* **adicionar(Tributo e)**: Es un método público de tipo void, el cual añade el objeto ingresado como parámetro a nuestro arreglo *aTributo*.
* **obtener(int pos)**: Es un método público de tipo Tributo, el cual devuelve un objeto de la clase Tributo, en base al valor entero ingresado como parámetro. El objeto a devolver será el correspondiente a la posición de dicho valor entero en nuestro arreglo.
* **getTamaño()**: Es un método público de tipo int, el cual devuelve el tamaño de nuestro arreglo.
* **eliminar(Tributo e)**: Es un método público de tipo void, el cual remueve el objeto ingresado como parámetro de nuestro arreglo *aTributo*.
* **cargar()**: Es un método público de tipo void, el cual lee los datos de nuestro archivo de texto correspondiente a primera categoría, y luego guarda dichos datos en nuestro arreglo.
* **grabar()**: Es un método público de tipo void, el cual se encarga de guardar los datos de nuestro arreglo en nuestro archivo de texto correspondiente.
* **Arreglo1Tributo():** Es el constructor que utilizamos para la clase Arreglo1Tributo, éste no posee parámetro alguno. Se encarga de inicializar el arreglo aTributo y utilizar el método *cargar*.

#### Arreglo2Tributo

* **aTributo:** Es un atributo privado Arraylist de objetos de tipo Tributo, el cual nos permite almacenar la información de los trámites de quinta categoría realizados.
* **adicionar(Tributo e)**: Es un método público de tipo void, el cual añade el objeto ingresado como parámetro a nuestro arreglo *aTributo*.
* **obtener(int pos)**: Es un método público de tipo Tributo, el cual devuelve un objeto de la clase Tributo, en base al valor entero ingresado como parámetro. El objeto a devolver será el correspondiente a la posición de dicho valor entero en nuestro arreglo.
* **getTamaño()**: Es un método público de tipo int, el cual devuelve el tamaño de nuestro arreglo.
* **eliminar(Tributo e)**: Es un método público de tipo void, el cual remueve el objeto ingresado como parámetro de nuestro arreglo *aTributo*.
* **cargar()**: Es un método público de tipo void, el cual lee los datos de nuestro archivo de texto correspondiente a segunta categoría, y luego guarda dichos datos en nuestro arreglo.
* **grabar()**: Es un método público de tipo void, el cual se encarga de guardar los datos de nuestro arreglo en nuestro archivo de texto correspondiente.
* **Arreglo2Tributo():** Es el constructor que utilizamos para la clase Arreglo2Tributo, éste no posee parámetro alguno. Se encarga de inicializar el arreglo aTributo y utilizar el método *cargar*.

#### Arreglo3Tributo

* **aTributo:** Es un atributo privado Arraylist de objetos de tipo Tributo, el cual nos permite almacenar la información de los trámites de quinta categoría realizados.
* **adicionar(Tributo e)**: Es un método público de tipo void, el cual añade el objeto ingresado como parámetro a nuestro arreglo *aTributo*.
* **obtener(int pos)**: Es un método público de tipo Tributo, el cual devuelve un objeto de la clase Tributo, en base al valor entero ingresado como parámetro. El objeto a devolver será el correspondiente a la posición de dicho valor entero en nuestro arreglo.
* **getTamaño()**: Es un método público de tipo int, el cual devuelve el tamaño de nuestro arreglo.
* **eliminar(Tributo e)**: Es un método público de tipo void, el cual remueve el objeto ingresado como parámetro de nuestro arreglo *aTributo*.
* **cargar()**: Es un método público de tipo void, el cual lee los datos de nuestro archivo de texto correspondiente a tercera categoría, y luego guarda dichos datos en nuestro arreglo.
* **grabar()**: Es un método público de tipo void, el cual se encarga de guardar los datos de nuestro arreglo en nuestro archivo de texto correspondiente.
* **Arreglo3Tributo():** Es el constructor que utilizamos para la clase Arreglo3Tributo, éste no posee parámetro alguno. Se encarga de inicializar el arreglo aTributo y utilizar el método *cargar*.

#### Arreglo4Tributo

* **aTributo:** Es un atributo privado Arraylist de objetos de tipo Tributo, el cual nos permite almacenar la información de los trámites de quinta categoría realizados.
* **adicionar(Tributo e)**: Es un método público de tipo void, el cual añade el objeto ingresado como parámetro a nuestro arreglo *aTributo*.
* **obtener(int pos)**: Es un método público de tipo Tributo, el cual devuelve un objeto de la clase Tributo, en base al valor entero ingresado como parámetro. El objeto a devolver será el correspondiente a la posición de dicho valor entero en nuestro arreglo.
* **getTamaño()**: Es un método público de tipo int, el cual devuelve el tamaño de nuestro arreglo.
* **eliminar(Tributo e)**: Es un método público de tipo void, el cual remueve el objeto ingresado como parámetro de nuestro arreglo *aTributo*.
* **cargar()**: Es un método público de tipo void, el cual lee los datos de nuestro archivo de texto correspondiente a cuarta categoría, y luego guarda dichos datos en nuestro arreglo.
* **grabar()**: Es un método público de tipo void, el cual se encarga de guardar los datos de nuestro arreglo en nuestro archivo de texto correspondiente.
* **Arreglo4Tributo():** Es el constructor que utilizamos para la clase Arreglo4Tributo, éste no posee parámetro alguno. Se encarga de inicializar el arreglo aTributo y utilizar el método *cargar*.

#### Arreglo5Tributo

* **aTributo:** Es un atributo privado Arraylist de objetos de tipo Tributo, el cual nos permite almacenar la información de los trámites de quinta categoría realizados.
* **adicionar(Tributo e)**: Es un método público de tipo void, el cual añade el objeto ingresado como parámetro a nuestro arreglo *aTributo*.
* **obtener(int pos)**: Es un método público de tipo Tributo, el cual devuelve un objeto de la clase Tributo, en base al valor entero ingresado como parámetro. El objeto a devolver será el correspondiente a la posición de dicho valor entero en nuestro arreglo.
* **getTamaño()**: Es un método público de tipo int, el cual devuelve el tamaño de nuestro arreglo.
* **eliminar(Tributo e)**: Es un método público de tipo void, el cual remueve el objeto ingresado como parámetro de nuestro arreglo *aTributo*.
* **cargar()**: Es un método público de tipo void, el cual lee los datos de nuestro archivo de texto correspondiente a quinta categoría, y luego guarda dichos datos en nuestro arreglo.
* **grabar()**: Es un método público de tipo void, el cual se encarga de guardar los datos de nuestro arreglo en nuestro archivo de texto correspondiente.
* **Arreglo5Tributo():** Es el constructor que utilizamos para la clase Arreglo5Tributo, éste no posee parámetro alguno. Se encarga de inicializar el arreglo aTributo y utilizar el método *cargar*.

#### ArregloRegistros

* **aTributo:** Es un atributo privado Arraylist de objetos de tipo Registro, el cual nos permite almacenar la información de los trámites de quinta categoría realizados.
* **adicionar(Registro e)**: Es un método público de tipo void, el cual añade el objeto ingresado como parámetro a nuestro arreglo *aRegistro*.
* **obtener(int pos)**: Es un método público de tipo Registro, el cual devuelve un objeto de la clase Tributo, en base al valor entero ingresado como parámetro. El objeto a devolver será el correspondiente a la posición de dicho valor entero en nuestro arreglo.
* **getTamaño()**: Es un método público de tipo int, el cual devuelve el tamaño de nuestro arreglo.
* **eliminar(Registro e)**: Es un método público de tipo void, el cual remueve el objeto ingresado como parámetro de nuestro arreglo *aRegistro*.
* **cargar()**: Es un método público de tipo void, el cual lee los datos de nuestro archivo de texto correspondiente a los registros de clientes, y luego guarda dichos datos en nuestro arreglo.
* **buscar(String dni)**: Es un método público de tipo Registro, el cual lee los datos de nuestro archivo de texto correspondiente a los registros de clientes, y luego compara el cada línea con nuestro parámetro de tipo String dni, para posteriormente devolver dicho valor si es que es igual al de nuestro parámetro.
* **buscarRucyContra(String ruc, String contra)**: Es un método público de tipo Registro, el cual lee los datos de nuestro archivo de texto correspondiente a los registros de clientes, y luego compara el cada línea con nuestro parámetro de tipo String ruc y String contra para posteriormente devolver dicho valor si es que es igual al de nuestro parámetro.
* **buscarRuc(String ruc)**: Es un método público de tipo Registro, el cual lee los datos de nuestro archivo de texto correspondiente a los registros de clientes, y luego compara el cada línea con nuestro parámetro de tipo String ruc para posteriormente devolver dicho valor si es que es igual al de nuestro parámetro.
* **grabar()**: Es un método público de tipo void, el cual se encarga de guardar los datos de nuestro arreglo en nuestro archivo de texto correspondiente.
* **ArregloRegistros():** Es el constructor que utilizamos para la clase ArregloRegistros, éste no posee parámetro alguno. Se encarga de inicializar el arreglo aRegistros y utilizar el método *cargar*.

### Captura de pantalla de un celular Descripción generada automáticamenteDiagrama de clases actualizado

### Librerías Gráficas

#### Concepto

Java nos ofrece las herramientas necesarias para que como desarrolladores podamos construir la mejor interfaz posible, utilizando todos los conceptos de programación orientada a objetos, de esta forma podemos aprovechar todo el poder del lenguaje. Una interfaz gráfica se compone de ventanas, iconos y acciones, con ello podemos construir un menú, o un elemento que reciba un texto o simplemente un botón que llame a alguna acción.

* **Librería Swing:**

Esta librería es una biblioteca de interfaces gráficas de usuario, mayormente conocidas como GUI. Dicha librería viene incluida en el entorno de desarrollo de Java (JDK) y , a su vez, es una extensión de otra librería gráfica llamada AWT.

* **Librería AWT: (Abstract Window Toolkit).**

Conjunto de herramientas GUI (Interfaz Gráfica con el Usuario) diseñadas para trabajar con múltiples plataformas, Interfaz de usuario, y sistema de ventanas independiente de la plataforma original de Lenguaje de programación Java. proporciona una serie de controles que podremos colocar en las aplicaciones visuales que implementemos. Dichos controles son subclases de la clase Component, y forman parte del paquete java.awt.

* **Componentes:**
* **JFrameForm**: Proporciona operaciones para manipular ventanas y contiene todos los componentes utilizados en la interfaz Gráfica de Usuario.
* **JInternalFrame**:Utilizada como Ventana interna dentro de un DesktopPane para implementar componentes, como un pequeño formulario dentro de otro.
* **Jpanel**: Componente con un layout manager propio, el cual podemos utilizar para almacenar otros componentes en un solo grupo y poder trasladarlos o implementarlos en otro formulario más fácilmente..
* **JButton**: Componente que permite implementar un Botón, el cual puede ser pulsado y posteriormente procederá a realizar una serie de instrucciones ya definidas en el código del formulario.
* **JLabel**: Componente utilizado usualmente para mostrar un pequeño mensaje de texto al usuario, aunque también puede ser utilizado para insertar imágenes en nuestro formulario.
* **JTextField**: Componente que permite al usuario ingresar una línea de texto, y poder recepcionar dicho arreglo de caracteres para utilizarlo en nuestro programa. Asimismo, se puede utilizar para mostrar una pequeña cantidad de información.
* **JTextArea**: Componente con una funcionalidad similar al JTextField, la principal diferencia radica en la cantidad de texto que se puede ingresar/mostrar, siendo el JTextArea la que permite mayor información.
* **JCheckBox**: Permite seleccionar o deseleccionar uno o varios ítems de un conjunto de opciones.
* **JRadioButton**: Componente para mostrar varias opciones de las cuales sólo se puede seleccionar una.Usualmente se utiliza este componente para el ingreso de datos como el género o estado civil.
* **JComboBox**: Componente con una funcionalidad similar al JRadioButton, la cual nos permite escoger una opción entre varias; siendo la principal diferencia la presentación de cada uno de estos componente.
* **JList**: Permite implementar una lista con varias opciones de las cuales se puede seleccionar una o varias.
* **JMenuBar**: Permite implementar un menú de ítems y sub ítems de opciones en java.
* **JTable**: Permite implementar una tabla para poder mostrar Información en sus celdas.
* **JFreeChart**: es una librería la cual nos permite hacer gráficos de una forma muy sencilla, esta librería posee todo tipo de gráficos, desde tortas, líneas, barras, etc.
* **Librería JavaFX:**

Es una familia de productos y tecnologías de Oracle Corporation, para la creación de Rich Internet Applications, esto es, aplicaciones web que tienen las características y capacidades de aplicaciones de escritorio, incluyendo aplicaciones multimedia interactivas.

#### Aplicación

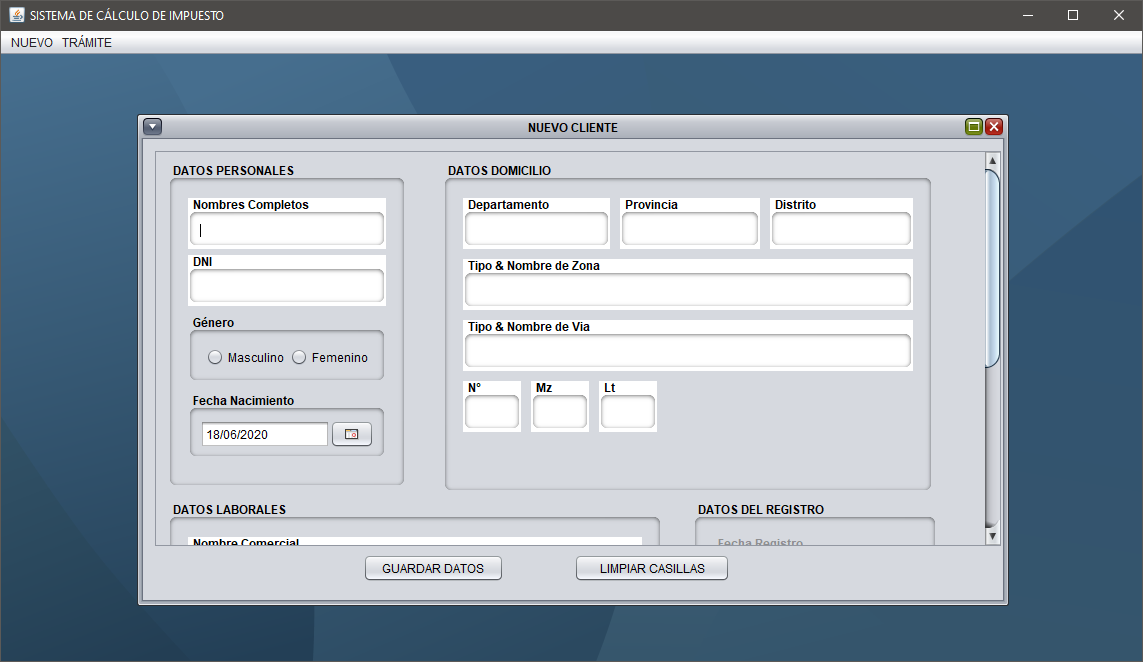


Ilustración Interfaz de registro de nuevo cliente

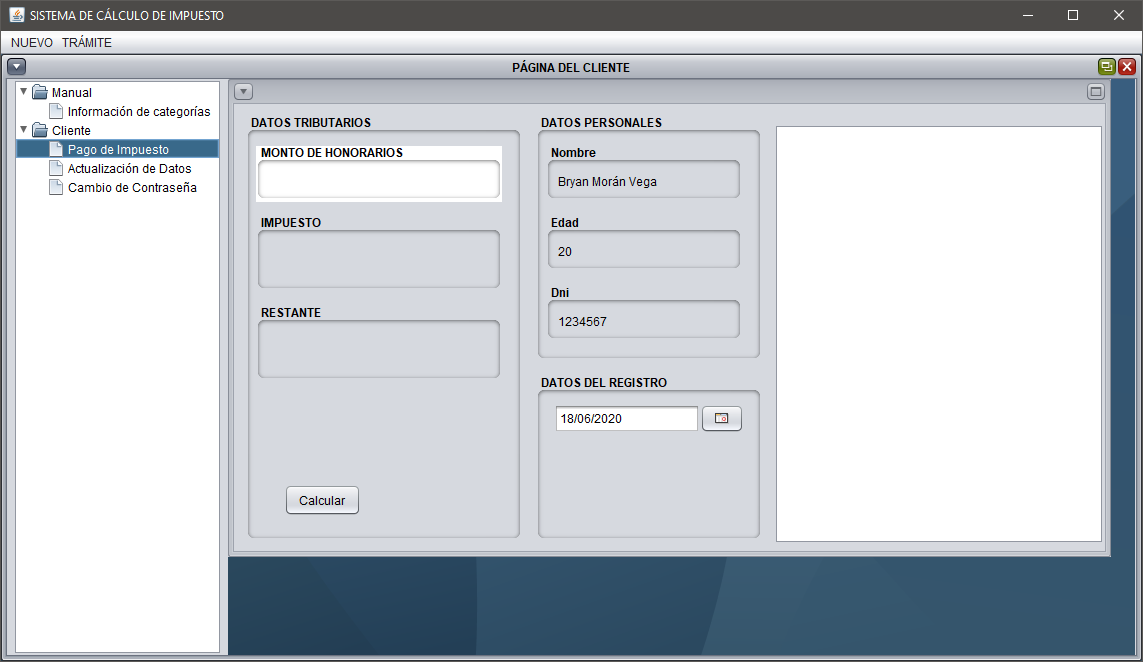


Ilustración Interfaz de pago de cuarta categoría

### Pruebas de software

#### Concepto

* **Unit testing o pruebas unitarias:** Es un nivel de prueba de software donde se prueban unidades / componentes individuales de un software. Son las pruebas de software más básicas, las cuales se encargan de comprobar el correcto funcionamiento de unidades o componentes individuales; siendo estos usualmente las clases y sus métodos
* **Pair programming o programación en pareja**: Consiste en que una persona desarrolla y otra se sienta a su lado para revisar el código en tiempo de escritura (es habitual que uno no vea sus propios errores). Cada periodo corto, se intercambian. Con esta técnica se consigue un código más sencillo, fácil de mantener y, gracias al hecho de que lo revise otra persona, con menos posibilidades de errores.
* **Pruebas Alpha**: Las pruebas alpha, del inglés alpha testing, son pruebas de software realizadas cuando el sistema está en desarrollo y cuyo objetivo es asegurar que lo que estamos desarrollando es probablemente correcto y útil para el cliente. Por ejemplo, podríamos desarrollar un prototipo de una parte del sistema que no estamos seguros de cómo implementar la interfaz. Si las pruebas alpha devuelven unos comentarios positivos entonces podríamos seguir por esa vía. Si devolvieran resultados muy negativos tendríamos que replantear el problema para adaptarse mejor a los requerimientos.
* **Pruebas Beta**: Las pruebas beta, del inglés beta testing, son las pruebas de software que se realizan cuando el sistema está teóricamente correcto y pasa a ejecutarse en un entorno real. Es la fase siguiente a las pruebas Alpha. No importa lo bueno que sea nuestro proceso de desarrollo, siempre habrá fallos que no han sido descubiertos por los desarrolladores ni por el equipo de pruebas. Las pruebas beta son pruebas para localizar esos problemas que no han sido detectados y poder corregirlos antes de liberar una versión. Debería ser realizada por usuarios finales. Dependiendo de la naturaleza del software podría, por ejemplo, ser realizada por compañeros de trabajo, por algunos clientes reales, o por una combinación de ambos.
* **Pruebas de Regresión**: Cualquier tipo de pruebas de software con el objeto de descubrir errores (bugs), carencias de funcionalidad, o divergencias funcionales con respecto al comportamiento esperado del software, causados por la realización de un cambio en el programa. Se evalúa el correcto funcionamiento del software desarrollado frente a evoluciones o cambios funcionales. El propósito de éstas es asegurar que los casos de prueba que ya habían sido probados y fueron exitosos permanezcan así.
* **Pruebas de Integración**: Las pruebas de integración verifican que los diferentes módulos y/o servicios usados por nuestra aplicación funcionen en armonía cuando trabajan en conjunto. Las pruebas de integración son típicamente el paso siguiente a las pruebas unitarias.
* **Pruebas Software no Funcionales** que incluyen las pruebas de: Rendimiento, Carga, Estrés, Usabilidad, Mantenibilidad, Fiabilidad o Portabilidad, entre otras. Por tanto se centran en características del software que establecen “cómo trabaja el sistema “. Las características no funcionales del software se pueden medir de diversas maneras, por ejemplo, por medio de tiempos de respuesta en el caso de pruebas de rendimiento o por número máximo de sesiones en pruebas de stress.
* **Pruebas Software Funcionales**. Típicamente encontraremos el comportamiento del sistema, subsistema o componente software descrito en especificaciones de requisitos o casos de uso, aunque también puede no estar documentado (“que funcione como el sistema al que sustituye”). Es decir, con las funciones establecemos “lo que el sistema hace”. Estas pruebas se definen a partir de funciones o características (como decimos, bien descritas en documentos o bien interpretadas por los probadores) y su interoperabilidad con sistemas específicos, pudiendo ejecutarse en todos los niveles de pruebas (componentes, integración, sistema, etc).
* **Prueba de stress**: esta evaluación pone a prueba la robustez y la confiabilidad del software sometiéndose a condiciones de uso extremas. El objetivo es saturar el programa hasta un punto de quiebre donde aparezcan bugs (defectos) potencialmente peligrosos.

#### Aplicación

Para nuestro proyecto, se tuvo en cuenta el uso de la prueba de stress, la cual gracias a la finalización de cada Sprint. Podemos verificar el funcionamiento de ciertas fases de la programación. Una de ellas es la validación de caracteres y su respectiva longitud. Además de eso, se aplicó la prueba unitaria; ayudándonos a verificar el correcto funcionamiento de las clases y métodos. Gracias al Done de cada Sprint, nos fijamos en que cada fase del proyecto cumpla con ciertos requisitos mínimos como para darlo por terminados.

# **CAPÍTULO III**

## Conclusiones y recomendaciones

### Conclusiones

Concluimos que nuestro proyecto sirve para calcular el impuesto a la renta de las 5 categoría siendo similar cada una de ellas.pero únicas al momento de ver las necesidades del usuario. A su vez descubrimos que hay infinidad de interfaces gráficas que podríamos implementar a nuestro proyecto para darle una vista más fácil de utilizar al usuario. A si mismo con este proyecto aplicamos todos los conocimientos con respectos  programación orientada a objetos principalmente al lenguaje de java lo cual nos permite tener una programación más estructurada a sí mismo de fácil manejo.También fue de suma importancia conocer e investigar sobre las 5 categorías de pago de impuesto que otorga lo que es la entidad de la sunat teniendo en cuenta el tipo de usuario.

### Recomendaciones

#### Para mejora del programa

* Se puede llegar a implementar un jcomboboxt que permita al cliente desactivar su estado de contribución.
* Se puede llegar a implementar un JtextField para que el área administrativa pueda realizar el ingreso de su nombre y corroborar con su respectiva clave de acceso.
* Se puede implementar una manera más exacta el cálculo de la edad del cliente dependiendo su fecha de nacimiento.
* Se puede implementar una interfaz gráfica de usuario más amigable y de fácil uso.
* Se puede implementar todos los casos pertenecientes a la segunda categoría.

#### Para una buena ejecución del Programa

* Windows: XP
* RAM: 1828M
* Espacio en disco: 124 MB para JDK; 2 MB para Java Update
* Procesador: Mínimo Pentium 2 a 266 MHz
* Microsoft SDK for Java
* Microsoft virtual machine for Java
* Oracle JDBC drivers

# **CAPÍTULO IV**

## Bibliografía y anexos

#### Bibliografía

* **Cálculo de renta de primera categoría según la SUNAT:**
  + <https://orientacion.sunat.gob.pe/index.php/personas-menu/impuesto-a-la-renta-personas-ultimo/rentas-de-primera-categoria-personas/3039-02-calculo-de-rentas-de-primera-categoria>
* **Cálculo de renta de segunda categoría por venta de valores mobiliarios según la SUNAT:**
  + <https://orientacion.sunat.gob.pe/index.php/personas-menu/impuesto-a-la-renta-personas-ultimo/rentas-de-segunda-categoria-personas/rentas-de-segunda-categoria-por-venta-de-valores-mobiliarios-personas/3062-02-determinacion-del-impuesto-por-venta-de-valores-mobliliarios>
* **Cálculo de renta de tercera categoría según la SUNAT:**
  + <https://renta.sunat.gob.pe/sites/default/files/inline-files/cartilla%20Instrucciones%20Empresa_2.pdf>
* **Cálculo de renta de cuarta categoría según la SUNAT:**
  + <https://orientacion.sunat.gob.pe/index.php/personas-menu/impuesto-a-la-renta-personas-ultimo/rentas-de-cuarta-categoria-personas/3064-02-calculo-del-impuesto-por-rentas-de-cuarta-categoria>
* **Cálculo de renta de quinta categoría según la SUNAT:**
  + <https://orientacion.sunat.gob.pe/index.php/personas-menu/impuesto-a-la-renta-personas-ultimo/rentas-de-quinta-categoria-personas/3071-02-calculo-del-impuesto>
* **Información con referente al ubigeo:**
  + <http://webinei.inei.gob.pe:8080/sisconcode/proyecto/index.htm?proyectoTitulo=UBIGEO&proyectoId=3>
* **Iconos utilizados a lo largo del programa:**
  + <https://www.flaticon.es/>
* **Metodología Scrum:**
  + <https://proyectosagiles.org/que-es-scrum/>
* **Colecciones en Java:**
  + <https://www.adictosaltrabajo.com/2015/09/25/introduccion-a-colecciones-en-java/>
* **Archivos en Java:**
  + <https://ingenieria.udistrital.edu.co/pluginfile.php/39198/mod_resource/content/1/7.%20Archivos%20en%20Java.pdf>
* **Librerias gráficas:**
  + Swing: <https://www.incanatoit.com/2015/06/libreria-swing-interfaces-gui-jframe.html>
  + AWT: <http://dis.um.es/~bmoros/Tutorial/parte13/cap13-1.html>
  + JavaFX: <https://medium.com/@osmandi/qu%C3%A9-es-javafx-41cfc327bdf2>
* **Pruebas de software:**
  + <https://programacionymas.com/blog/tipos-de-testing-en-desarrollo-de-software>

#### Anexos

* **Librería RestrictedTextField:**

Esta librería fue utilizada para limitar el número de caracteres permitido en un textfield.Así mismo, nos permite restringir al textfield de sólo recibir un tipo de carácter específico, o ciertos caracteres particulares.

Enlace: <http://atxy2k.github.io/RestrictedTextField/>

* **Librería JCalendar:**

Esta librería fue utilizada para recibir la fecha ingresada por el usuario, utilizando el JDateChooser. A su vez, también nos permite la utilización del JSpinField para el ingreso de datos numéricos, como la cantidad de meses faltantes en quinta categoría.

Enlace: <https://toedter.com/jcalendar/>