PROPxCEL

2a ENTREGA

PROP

Subgrupo 1.4

Pérez Barroso, David (david.perez.barroso@estudiantat.upc.edu)
Risueño Martín, Iván (ivan.risueno@estudiantat.upc.edu)
Torra Garcia, Joaquim (joaquim.torra@estudiantat.upc.edu)

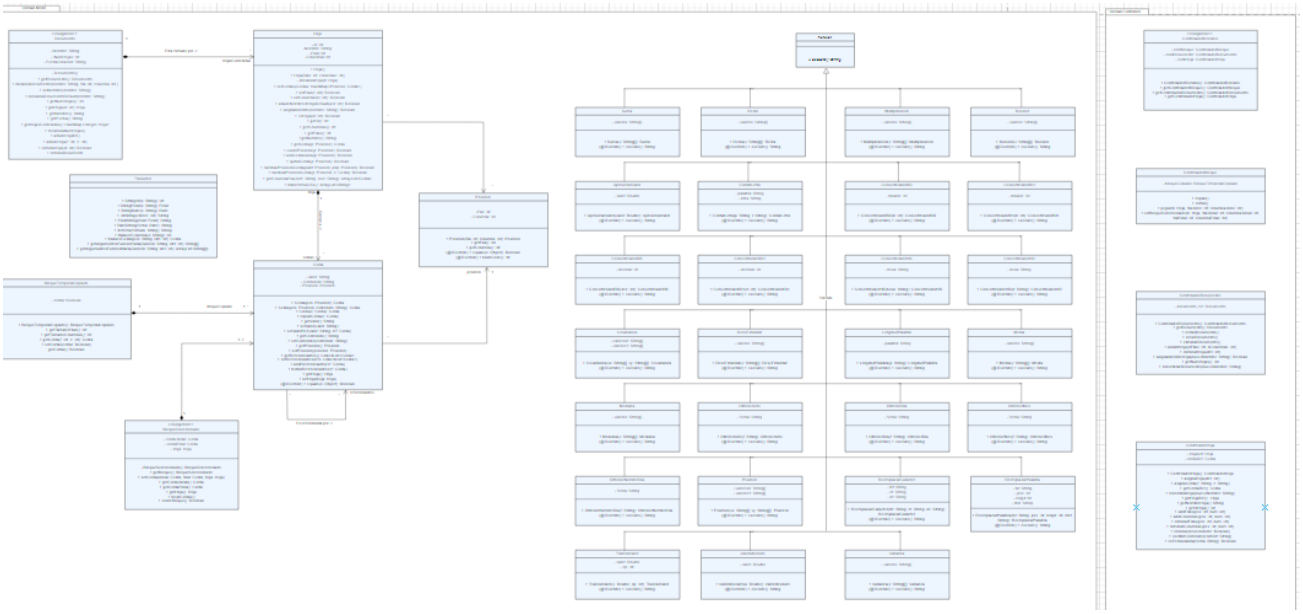
2a Entrega - 23 de Mayo de 2022
Versión del entregable: 2.1

Índice

1. Diagrama de clases y controladores del Dominio

1.1 Diagrama del modelo conceptual

Dado que no es posible poner el diagrama de tal manera que sea legible, se adjunta el archivo junto a este documento.



Restricciones textuales:

1. **Claves externas**: (Documento, Nombre), (Hoja, Id), (Celda, Posicion::Fila+Posicion::Columna+Hoja::Id), (Bloque, CeldaInicial + CeldaFinal), (Función, Id)

2. **Clave débil:** En una misma hoja no pueden haber dos Filas o dos Columnas con los mismos IDs, respectivamente.

3. **Clave débil**: En un mismo documento no pueden haber dos Hojas con los mismos IDs.

4. El ID de una Columna pertenece al rango ["A", "Z...Z"].

5. En Bloque, CeldaInicial <= CeldaFinal

6. En Hoja, Filas > 0 y Columnas > 0

7. Los atributos de una Función deben ser tipos de datos o referencias a celdas de tipo "An"

1.2 Explicación de las clases del Dominio

1.2.1 Documento

La clase Documento es la clase encargada de almacenar y gestionar todos los datos relacionados con un documento, es decir las hojas que contiene, su nombre y la fecha en el que lo crearon.

Dado que solo se contempla tener un documento a la vez, esta clase pasa a ser un singleton para satisfacer que haya solo una instancia de esta cuando el programa esté en funcionamiento. Puesto que la clase es singleton, su creadora tiene que ser privada y se accederá a una instancia de esta clase mediante una función pública que la devuelve. La función comprueba si existe una instancia, en caso afirmativo la devuelve, de lo contrario la crea y seguidamente la devuelve.

Sin embargo, dado que nosotros no contemplamos un documento vacío (sin hojas) y esta creadora no inicializa el documento, la clase contendrá un método el cual inicialice el documento con el nombre que le pasen como parámetro, con la fecha actual y con una hoja con tantas filas y columnas como le sean introducidos por parámetro (Por defecto serán 50 filas y 50 columnas).

Para poder almacenar documentos, haremos que esta clase sea *Serializable*, cosa que facilitará el trabajo a la hora de comunicarnos con la capa de persistencia.
Podremos enviar un objeto “documento” y este se traducirá automáticamente a datos de tipo byte.
Dado que documento contiene otra clase propia com lo es Hoja, para poder serializar Documento, necesitaremos que Hoja sea serializable también, a su vez, como Hoja contiene las clases propias Posicion y Celda, estas también tendrán que ser serializables con el mismo fin.

Cabe destacar que estas clases tendrán un atributo estático que contendrá un serialVersionUID personalizado para el correcto funcionamiento de la serialización.

1.2.1.1 Descripción de los métodos de la clase

- **inicializaDocumento**: Esta funcionalidad se encarga de inicializar un documento con el nombre pasado como parámetro, y le añade un Hoja con el número de filas y columnas pasados como parámetros. También da valor al atributo fechaCreación añadiendo la fecha actual de formato dd-mm-yyyy.

- **getHoja:** Este método devuelve una Hoja identificada por el identificador introducido que contiene el documento. Necesariamente la Hoja tiene que existir en el documento, de lo contrario esta función devuelve nulo.

- **recalculaNumHojas:** Esta función es la encargada de de recalcular el número de hojas que contiene el documento, básicamente comprueba el tamaño de la estructura de datos y actualiza el atributo.

- **añadeHoja**: Este método se encarga de añadir una hoja con tantas filas y columnas como indican los números introducidos. Además le asigna un identificador a la hoja creada basado en la posición que le corresponde a la hoja en la estructura de datos y actualiza el número de hojas contenidas del documento.

- **eliminaHoja:** Se encarga de eliminar una hoja que contiene el documento. Si la hoja es contenida y se borra correctamente la función devolverà cierto, de lo contrario, devolverá falso.

- **eliminaDocumento:** Esta función se encarga de eliminar la instancia de documento así como todos sus atributos.

- **leeCSV**: Añade una hoja al documento a partir de un ArrayList de String que le pasan como parámetro. Este ArrayList contiene los datos de una hoja en csv.

1.2.1.2 Descripción de los atributos

- **nombre**: Nombre del documento.

- **numHojas**: Número de hojas contenidas en el documento.

- **fechaCreacion:** Fecha de creación del documento.

- **hojasContenidas:** Hashmap de clave Integer y valor Hoja que contiene para cada Hoja existente, un identificador. Nótese que numHojas es un atributo derivado calculado.

1.2.2 Hoja

La clase Hoja es la clase encargada de almacenar y gestionar todos los datos relacionados con una hoja, es decir las filas y columnas que contiene, su nombre e identificador y las celdas y posiciones contenidas por esta.

La creadora añadirá las filas y columnas pasadas como parámetros, además inicializará la hoja mediante una función privada añadiendo tantas posiciones y celdas vacías como filas y columnas se hayan introducido.

A diferencia de los atributos inicializados con la creadora, tanto el nombre como el identificador serán asignados automáticamente con unos valores por defecto mediante una función pública que contiene la clase. Esta será llamada desde la clase documento que le asignará un identificador dependiendo del sitio que le corresponda a la hoja en la estructura de datos de documento donde las almacena. El nombre por defecto será “Hoja + su identificador”.

1.2.2.1 Descripción de los métodos de la clase

- **inicializaHoja**: Función privada llamada desde la creadora de hoja que inicializa la estructura de datos llenándola con tantas celdas y posiciones como indica el número de filas y columnas de la hoja.

- **añadeNombreIdHojaDefault:** Este método se encarga de añadir un identificador a la hoja mediante el número introducido y de asignarle un nombre por defecto el cual es “Hoja” más su identificador, p.e: “Hoja1”

- **esHoja:** Método que comprueba que la hoja sea la misma que la que tiene el identificador que le han pasado como parámetro. Devuelve cierto si lo es, falso de lo contrario.

- **getCelda:** Esta función se encarga de devolver una celda contenida en la hoja en la posición introducida. Para ello buscará en la estructura de datos si existe la posición introducida, en caso de ser cierto cogerá la celda que se encuentra en esa posición y la devolverá, de lo contrario devolverá nulo.

- **existePosicion:** Comprueba que la posición introducida existe en la hoja, en caso afirmativo devuelve cierto, de lo contrario falso.

- **addCeldaVacia:** Este método es el encargado de añadir una celda en la

hoja en la posición pasada como parámetro. Para ello se comprueba que la posición introducida no existe, en caso de no existir se crea una celda vacía con esa posición y se añade en la estructura de datos en la misma posición, seguidamente si todo ha salido correctamente devuelve cierto, de lo contrario si la posición ya existe se devuelve falso.

- **quitarCelda:** Borra una celda en la posición introducida como parámetro. Para ello comprueba que dicha posición exista en la hoja, de ser así, elimina la celda que se encuentra en esa posición junto a la posición.

  Seguidamente se comprueba si la posición existe en la estructura de datos y devolverá cierto si no existe. Tanto si al principio la posición no existe como si no se ha borrado correctamente se devolverá falso.

- **cambiarPosicionCelda:** Esta función se encarga de cambiar una celda de una posición dada, a otra posición dada. Para ello se comprueba que las dos posiciones introducidas existen en la estructura de datos. De ser así se cambia la celda de posición a la nueva posición asignándole todas las referencias que había en la nueva posición. En la posición anterior quedará una celda vacía.

  Seguidamente se comprueba que el cambio se ha efectuado correctamente, de ser así se devolverá cierto. En caso de que alguna de las posiciones introducidas no existan o no se haya efectuado el cambio de posición correctamente, devolverá falso.

  Está función además tiene una variante con el mismo objetivo solo que en la variante en vez de pasarle dos posiciones como parámetros, se le pasa la posición futura y la celda a mover.

- **getColumnaFila:** Este método se encarga de devolver una lista de celdas, contenidas en una fila o en una columna dependiendo del orden en los parámetros que se le pasan. Podrá devolver una columna recorriendola de arriba hacia abajo o de abajo hacia arriba, igual pasa con las filas, podrá recorrerlas de derecha a izquierda o de izquierda a derecha.

  P.e: A1, A3 devolverà las celdas que se encuentran en la fila que empieza en la posición 1,1 a la posición 1,3 leyéndolas de izquierda a derecha, si los parámetros se pasan al revés, las leerá de derecha a izquierda.

- **transformaCSV:** Transforma todo el contenido de la hoja en un ArrayList de Strings con los valores de las celdas de una misma fila separados por ";" y separación de filas con un espacio en blanco. Transformando así la hoja en formato CSV.

- **csvAHoja:** Transforma el ArrayList de string que le pasan, que contiene datos de una hoja en csv a un objeto Hoja y la devuelve.

1.2.2.2 Descripción de los atributos

- **id:** Identificador de la hoja en el documento.

- **nombre:** Nombre de la hoja.

- **filas:** Número de filas que contiene la hoja.

- **columnas:** Número de columnas que contiene la hoja.

- **celdas:** *Hashmap* cuyas claves son objetos de tipo Posicion y sus valores son de tipo Celda. Es el atributo que representa la hoja de cálculo como tal, donde se encuentran todas sus celdas.

1.2.3 Celda

La clase Celda representa una celda de la hoja de cálculo. Es la unidad mínima de información tanto del documento como de la hoja. En esta clase se almacena el contenido y valor que el usuario escribe en una celda.

En el caso que el usuario escriba una función en la celda, el contenido de esta será el texto de la función como por ejemplo “=abs(-2)” y el valor pasará a ser el resultado de la función, siguiendo el ejemplo anterior “2”. Sin embargo si el usuario escribe cualquier otro texto dentro de la celda que no sea ninguna función o referencia, valor y contenido tendrán el mismo valor.

Para poder realizar distintas operaciones en la que se necesita la posición de la celda sin necesidad de estar accediendo constantemente a la estructura de datos de la hoja, se guardará su posición. Además dispondrá de una lista con todas las celdas que tienen referencia a ella misma con tal de poder gestionar las referenciantes.

Dispondremos de tres creadoras con diferentes parámetros de entrada. La primera creadora se pasa una posición por parámetro y se inicializa valor y contenido a un string vacío y la posición a la posición pasada por parámetro.

La segunda variante, tendrá una posición y un string como parámetros de entrada e inicializará su posición y contenido con estos.

Finalmente en su tercera variante se le pasa una celda como parámetro, el objetivo de esta variante es crear una celda copia de la que se le pasa como parámetro, inicializando así el contenido, valor y posición con los de la celda pasada.

1.2.3.1 Descripción de los métodos de la clase

- **copiarCelda:** Esta función se encarga de copiar el contenido y el valor de la celda que le pasan como parámetro.

- **setValor**: Asigna un valor a la celda y a todas las que la referencian.

- **addReferenciante:** Este método se encarga de añadir una celda referenciante que le pasan como parámetro a otra celda.

- **borrarReferenciante:** Esta funcionalidad elimina la celda pasada como parámetro del conjunto de celdas referenciantes de la celda.

- **equals**: Se hace un *Override* de la función equals para que a la hora de comparar dos objetos Celda en alguna estructura de datos o en alguna condición, esta se pueda llevar a cabo correctamente.
  Se comparan principalmente por el objeto en sí y por su posición.

1.2.3.2 Descripción de los atributos

- **valor**: Valor mostrado en la interfaz gráfica del programa. Puede ser el resultado de una función, el valor propio de una celda referenciada o un valor en concreto especificado por el usuario.

- **contenido**: Contenido introducido por el usuario desde la interfaz gráfica del programa. Este contenido se guarda internamente y se muestra en las casillas que representan celdas el valor de las propias celdas, mientras que en el campo de texto *Fx* se muestra este atributo para que el usuario pueda editarlo libremente.

- **posicion**: Posicion de la celda en la hoja.

- **referenciantes**: LinkedList de objetos de tipo Celda usado para guardar las celdas que contienen una referencia a la celda *this*. Con esto conseguimos actualizar los valores de esas celdas cuando el contenido de la celda *this*(referenciada) cambie.

- **hoja**: Hoja en la cual se encuentra la celda.

1.2.4 Posicion

Posicion es la clase que representa la posición de una celda en la hoja. Está compuesta por dos enteros que representan el número de fila y el número de columna.

Cabe destacar que dado que esta clase se utiliza en la estructura de datos de la hoja (HashMap) como clave para almacenar las celdas, será necesario hacer override de los métodos equals y hashCode para el correcto funcionamiento.

1.2.4.1 Descripción de los métodos de la clase

- **equals**: Se hace un *Override* de la función equals para que a la hora de comparar dos objetos Posicion en alguna estructura de datos o en alguna condición, esta se pueda llevar a cabo correctamente.
  Se comparan principalmente por el objeto en sí y por su fila y columna.

- **hashCode:** Se hace un *Override* de la función hashCode para poder incluir esta clase como clave en una estructura Hash. Concretamente en la estructura de datos de Hoja.

1.2.4.2 Descripción de los atributos

- **fila:** Fila de la posición en cuestión (empieza en 1).

- **columna:** Columna de la posición en cuestión (empieza en 1).

1.2.5 Traductor

La clase Traductor es la clase dedicada a la conversión de los diferentes tipos de datos y a la detección de texto y argumentos escritos por el usuario en las diferentes celdas.
Dado que no es necesario ninguna instanciación de esta clase, la misma no tiene constructora y sus métodos son estáticos.

La funcionalidad principal de esta clase consiste en comunicar al resto de clases del dominio de si el usuario está escribiendo una función y cuál de ellas, para que el resto de clases puedan llamar a los procesos necesarios y cumplir los deseos del usuario.

Otra de las funcionalidades a destacar es que esta clase tiene varios métodos para obtener los argumentos de las diferentes funciones escritas por el usuario, para que se puedan realizar en otras clases los cálculos correctamente.

Finalmente destacar la funcionalidad de traducir una posición de celda del tipo letra-numero (“A1”) a una posición de dos enteros (1-1), dado que el resto de clases trabajan con posiciones con números enteros.

1.2.5.1 Descripción de los métodos de la clase

- **StringInt:** Hace un parse de String a entero con control de excepciones.

- **StringFloat:** Hace un parse de String a Float con control de excepciones.

- **StringDate:** Hace un parse de String a Date con control de excepciones.

- **IntString:** Hace un parse de entero a String.

- **FloatString:** Hace un parse de Float a String.

- **DateString:** Hace un parse de Date a String.

- **detecta:** Esta función se encarga de devolver un código que indica el tipo de función o valor del texto introducido como parámetro. Las demás clases podrán llamarla con el fin de saber qué tipo de función o dato es lo que ha escrito el usuario en una celda, para más tarde poder hacer las operaciones correspondientes.

  Algunos de los código devueltos son: Si el texto son funciones, por ejemplo "=suma()" el código será "#SUMA", si en cambio es "=aprox()" devolverá "#APROX", si ahora el texto es una referencia del tipo "=$A1" devolverá "#REFERENCIA", si es un valor corriente devuelve "#VALUE". Finalmente si se escribe una función con algun error gramatical, devuelve "#ERRORFUNC".

- **traduceColumna:** Este método es el encargado de traducir el texto introducido como parámetro en nomenclatura de columnas de A - ZZ a su valor numérico. Devuelve un entero con el valor de la posición de la columna, comenzando por 1.

- **traduceCelda:** Este método devuelve una Celda de la hoja actual dada su posición en string de tipo "A1" o "$A1".

- **getArgumentosFuncion1aria:** Este método devuelve un array de strings correspondiente a los argumentos de una funcion unária, las cuales son una referencia, una lista de referencias("$A1:$B2"), funciones como la de valor absoluto o un valor en concreto.

- **getArgumentosFuncionNaria:** Este metodo devuelve un ArrayList<String[]> con los argumentos transformados a valores de una función del estilo "=func(arg1, arg2)".

1.2.6 BloqueSeleccionado

La clase BloqueSeleccionado sirve principalmente para almacenar un conjunto de celdas seleccionadas.

Dispone de una celda inicial la cual será la celda superior izquierda que marca el

principio del bloque y de una celda final refiriéndose a la celda inferior derecha que marca el final del bloque.

Dado que pueden haber varias hojas a la vez en el mismo documento, necesitaremos también almacenar la hoja a la que nos referimos para poder operar correctamente.

1.2.6.1 Descripción de los métodos de la clase

- **getBloque:** Este método retorna el bloque seleccionado actual, ya que es una clase singleton.

- **setCelda:** Este método setea las celdas que forman los extremos superior izquierdo e inferior derecho del bloque seleccionado.

- **getCeldaInicial:** Retorna la celda superior izquierda.

- **getCeldaFinal:** Retorna la celda inferior derecha.

- **clearCeldas:** Borra el bloque seleccionado.

- **existeBloque:** Devuelve true si existe un bloque seleccionado, false en caso contrario.

1.2.6.2 Descripción de los atributos

- **celdaInicial:** Celda correspondiente a la esquina superior izquierda del bloque.

- **celdaFinal:** Celda correspondiente a la esquina inferior derecha del bloque.

- **hoja:** Hoja donde se encuentran estas celdas.

1.2.7 BloqueTemporalCopiado

La clase BloqueTemporalCopiado será la clase que nos servirá principalmente de portapapeles.

Su función principal es tener un bloque de celdas copiado y dado que sólo

contemplamos poder copiar un bloque a la vez, es necesario que esta clase tenga solo una instancia.

Para saber si un bloque ha sido copiado con la funcionalidad de cortar, esta clase almacena un atributo boolean que indicará cierto si se cumple lo comentado, y falso de lo contrario.

1.2.7.1 Descripción de los métodos de la clase

- **BloqueTemporalCopiado:** Incluimos la constructora porque se encarga de adquirir las celdas que forman el bloque seleccionado actual y se guarda todo el bloque seleccionado, incluyendo las celdas intermedias. Esto es así porque se debe poder copiar celdas que ya no están seleccionadas.

- **getTamanoFilas:** Devuelve el tamaño de filas que se han copiado o cortado.

- **getTamanoColumnas:** Devuelve el tamaño de columnas que se han copiado or cortado.

- **getCelda:** Dadas la fila y la columna relativa al bloque, devuelve la celda que se encuentra en esa posición.

1.2.7.2 Descripción de los atributos

- **bloqueCopiado:** Matriz de celdas que contiene todas las celdas que forman el bloque previamente seleccionado y posteriormente copiado o cortado.

- **cortar:** Booleano que nos indicará si hace falta borrar la instancia de bloqueTemporalCopiado cuando peguemos el contenido.

1.2.8 Funcion

La clase Función es una clase abstracta de la cual es padre de todas las subclases que definen todos los tipos de funciones que contempla nuestro sistema.

Posee un único método abstracto que será redefinido en cada hijo para ejecutar la función correspondiente.

1.2.8.1 Descripción de los métodos de la clase

- **execute:** Es una función abstracta que devuelve un String. Esta función será sobreescrita por todas las clases hijas de esta.

1.2.9 AproximarValor

La clase AproximarValor es una subclase de la clase Funcion, encargada de gestionar la función que aproxima un valor numérico.

Tiene como atributo un parámetro *double* privado llamado 'valor' que representa el valor a aproximar.

1.2.9.1 Descripción de los métodos de la clase

- **AproximarValor**: Creadora de la clase a la cual le pasan un valor *double* por parámetro y lo almacena en el atributo privado valor.

- **execute**: Hace *Override* de la función abstracta de su padre y aproxima el número que tiene almacenado en el atributo privado de la clase, devolviendo este valor aproximado en un string.

1.2.9.2 Descripción de los atributos

- **valor:** Valor a aproximar.

1.2.10 ValorAbsoluto

La clase ValorAbsoluto es una subclase de la clase Funcion, encargada de gestionar la función que transforma un valor numérico en valor Absoluto.

Tiene como atributo un parámetro *double* privado llamado 'valor' que representa el valor a transformar en su valor absoluto.

1.2.10.1 Descripción de los métodos de la clase

- **ValorAbsoluto**: Creadora de la clase a la cual le pasan un valor *double* por parámetro y lo almacena en el atributo privado valor.

- **execute**: Hace *Override* de la función abstracta de su padre y ejecuta el valor absoluto del número que tiene almacenado en el atributo privado de la clase, devolviendo este valor en valor absoluto en un string.

1.2.10.2 Descripción de los atributos

- **valor:** Valor necesario ejecutar la función.

1.2.11 TruncarValor

La clase TruncarValor es una subclase de la clase Funcion, encargada de gestionar la función que trunca un valor numérico.

Tiene como atributo un parámetro *double* privado llamado 'valor' que representa el valor a truncar y un atributo privado *int* llamado 'op' que representa el número de decimales que quieres dejar una vez truncado.

1.2.11.1 Descripción de los métodos de la clase

- **TruncarValor**: Creadora de la clase a la cual le pasan un valor *double* y un valor *int* por parámetro y los almacena en los atributos privados valor y op.

- **execute**: Hace *Override* de la función abstracta de su padre y trunca el número que tiene almacenado en 'valor' (atributo privado de la clase) a tantos números como dice 'op' (atributo privado de la clase), devolviendo el valor truncado en un string.

1.2.11.2 Descripción de los atributos

- **valor:** Valor a truncar.

- **op:** Posición del decimal a truncar el número.

1.2.12 ConvertirValorBD

La clase ConvertirValorBD es una subclase de la clase Funcion, encargada de gestionar la función que transforma un valor en binario a un valor en decimal.

Tiene como atributo un parámetro *integer* privado llamado 'binario' que representa el valor binario a transformar en decimal.

1.2.12.1 Descripción de los métodos de la clase

- **ConvertirValorBD**: Creadora de la clase a la cual le pasan un valor *int* por parámetro y lo almacena en el atributo privado binario.

- **execute**: Hace *Override* de la función abstracta de su padre y transforma el número que tiene almacenado en 'binario' (atributo privado de la clase) a un valor decimal, devolviendo el valor decimal en un string.

1.2.12.2 Descripción de los atributos

- **Binario:** Binario a convertir a decimal.

1.2.13 ConvertirValorBH

La clase ConvertirValorBH es una subclase de la clase Funcion, encargada de gestionar la función que transforma un valor en binario a un valor en hexadecimal.

Tiene como atributo un parámetro *integer* privado llamado 'binario' que representa el valor binario a transformar en hexadecimal.

1.2.13.1 Descripción de los métodos de la clase

- **ConvertirValorBH**: Creadora de la clase a la cual le pasan un valor *int* por parámetro y lo almacena en el atributo privado binario.

- **execute**: Hace *Override* de la función abstracta de su padre y transforma el número que tiene almacenado en 'binario' (atributo privado de la clase) a un valor hexadecimal, devolviendo el valor hexadecimal en un string.

1.2.13.2 Descripción de los atributos

- **Binario:** Binario a convertir a hexadecimal.

1.2.14 ConvertirValorDB

La clase ConvertirValorDB es una subclase de la clase Funcion, encargada de gestionar la función que transforma un valor en decimal a un valor en binario.

Tiene como atributo un parámetro *integer* privado llamado 'decimal' que representa el valor decimal a transformar en binario.

1.2.14.1 Descripción de los métodos de la clase

- **ConvertirValorDB**: Creadora de la clase a la cual le pasan un valor *int* por parámetro y lo almacena en el atributo privado decimal.

- **execute**: Hace *Override* de la función abstracta de su padre y transforma el número que tiene almacenado en 'decimal' (atributo privado de la clase) a un valor binario, devolviendo el valor binario en un string.

1.2.14.2 Descripción de los atributos

- **Decimal:** Decimal a convertir a binario.

1.2.15 ConvertirValorDH

La clase ConvertirValorDH es una subclase de la clase Funcion, encargada de gestionar la función que que transforma un valor en decimal a un valor en hexadecimal.

Tiene como atributo un parámetro *integer* privado llamado 'decimal' que representa el valor decimal a transformar en hexadecimal.

1.2.15.1 Descripción de los métodos de la clase

- **ConvertirValorDH**: Creadora de la clase a la cual le pasan un valor *int* por parámetro y lo almacena en el atributo privado decimal.

- **execute**: Hace *Override* de la función abstracta de su padre y transforma el número que tiene almacenado en 'decimal' (atributo privado de la clase) a un valor hexadecimal, devolviendo el valor hexadecimal en un string.

1.2.15.2 Descripción de los atributos

- **Decimal:** Decimal a convertir a valor hexadecimal.

1.2.16 ConvertirValorHB

La clase ConvertirValorHD es una subclase de la clase Funcion, encargada de gestionar la función que que transforma un valor en hexadecimal a un valor en binario.

Tiene como atributo un parámetro *string* privado llamado 'hexa' que representa el valor hexadecimal a transformar en binario.

1.2.16.1 Descripción de los métodos de la clase

- **ConvertirValorHB**: Creadora de la clase a la cual le pasan un valor *string* por parámetro y lo almacena en el atributo privado hexa.

- **execute**: Hace *Override* de la función abstracta de su padre y transforma el número que tiene almacenado en 'hexa' (atributo privado de la clase) a un valor binario, devolviendo el valor binario en un string.

1.2.16.2 Descripción de los atributos

- **hexa:** Valor hexadecimal a convertir a binario.

1.2.17 ConvertirValorHD

La clase ConvertirValorHD es una subclase de la clase Funcion, encargada de gestionar la función que que transforma un valor en hexadecimal a un valor en decimal.

Tiene como atributo un parámetro *string* privado llamado 'hexa' que representa el valor hexadecimal a transformar en decimal.

1.2.17.1 Descripción de los métodos de la clase

- **ConvertirValorHD**: Creadora de la clase a la cual le pasan un valor *string* por parámetro y lo almacena en el atributo privado hexa.

- **execute**: Hace *Override* de la función abstracta de su padre y transforma el número que tiene almacenado en 'hexa' (atributo privado de la clase) a un valor decimal, devolviendo el valor decimal en un string.

1.2.17.2 Descripción de los atributos

- **hexa:** Valor hexadecimal a convertir a decimal.

1.2.18 ObtenerAño

La clase ObtenerAño es una subclase de la clase Funcion, encargada de gestionar la función que obtiene el año de una fecha.

Tiene como atributo un parámetro *string* privado llamado 'fecha' que representa la fecha de la cual se obtendrá el año.

1.2.18.1 Descripción de los métodos de la clase

- **ObtenerAño**: Creadora de la clase a la cual le pasan un valor *string* por parámetro y lo almacena en el atributo privado fecha.

- **execute**: Hace *Override* de la función abstracta de su padre y coge el año de la fecha almacenada en el atributo privado de la clase y lo devuelve en un *string*. Si el año es un número negativo (fecha errónea), devuelve "#ERROR".

1.2.18.2 Descripción de los atributos

- **fecha:** Fecha de la cual extraer el año.

1.2.19 ObtenerDia

La clase ObtenerDia es una subclase de la clase Funcion, encargada de gestionar la función que obtiene el día de una fecha.

Tiene como atributo un parámetro *string* privado llamado 'fecha' que representa la fecha de la cual se obtendrá el día.

1.2.19.1 Descripción de los métodos de la clase

- **ObtenerDia**: Creadora de la clase a la cual le pasan un valor *string* por parámetro y lo almacena en el atributo privado fecha.

- **execute**: Hace *Override* de la función abstracta de su padre y coge el día de la fecha almacenada en el atributo privado de la clase y lo devuelve en un *string*. Si el dia es un número negativo (fecha errónea), devuelve "#ERROR".

1.2.19.2 Descripción de los atributos

- **fecha:** Fecha de la cual extraer el día.

1.2.20 ObtenerMes

La clase ObtenerMes es una subclase de la clase Funcion, encargada de gestionar la función que obtiene el mesde una fecha.

Tiene como atributo un parámetro *string* privado llamado 'fecha' que representa la fecha de la cual se obtendrá el mes.

1.2.20.1 Descripción de los métodos de la clase

- **ObtenerMes**: Creadora de la clase a la cual le pasan un valor *string* por parámetro y lo almacena en el atributo privado fecha.

- **execute**: Hace *Override* de la función abstracta de su padre y coge el mes de la fecha almacenada en el atributo privado de la clase, selecciona el nombre del mes (almacenados en un vector) asignado al número y lo devuelve en un *string*. Si el mes no se encuentra entre un número del 1 al 12 (fecha errónea), devuelve "#ERROR".

1.2.20.2 Descripción de los atributos

- **fecha:** Fecha de la cual extraer el mes.

1.2.21 ObtenerNombreDia

La clase ObtenerNombreDia es una subclase de la clase Funcion, encargada de gestionar la función que obtiene el nombre del día de una fecha.

Tiene como atributo un parámetro *string* privado llamado 'fecha' que representa la fecha de la cual se obtendrá el día.

1.2.21.1 Descripción de los métodos de la clase

- **ObtenerNombreDia**: Creadora de la clase a la cual le pasan un valor *string* por parámetro y lo almacena en el atributo privado fecha.

- **execute**: Hace *Override* de la función abstracta de su padre y coge el dia de la fecha almacenada en el atributo privado de la clase, mediante la función *DayOfWeek* transformaremos el número del día al nombre del dia de la semana, el problema es que nos devuelve el nombre en inglés, y lo queremos en castellano, por lo cual haremos un getValue() que nos devolverá el número que corresponde al nombre del dia de la semana y mediante un diccionario (HashMap) lo cambiaremos al castellano, seguidamente devolveremos este nombre. Si alguno de los componentes es erróneo o no es una fecha valida (fecha errónea), devuelve "#ERROR" o "#FECHA_NO_VALIDA", según el caso.

1.2.21.2 Descripción de los atributos

- **fecha:** Fecha de la cual extraer el nombre del día.

1.2.22 ContarLetra

La clase ContarLetra es una subclase de la clase Funcion, encargada de gestionar la función que obtiene el número de veces que aparece una letra en una palabra o frase.

Tiene como atributo un parámetro *string* privado llamado 'palabra' que representa la palabra o frase de la cual se contará la letra y otro atributo privado *string* 'letra' que representa la letra a contar.

1.2.22.1 Descripción de los métodos de la clase

- **ContarLetra**: Creadora de la clase a la cual le pasan dos valores *string* por parámetro y los almacena en los atributos privados. El primero corresponde con la palabra o frase y el segundo con la letra a contar.

- **execute**: Hace *Override* de la función abstracta de su padre, en ella se utilizaran los dos atributos privados de la clase para realizar la funcionalidad.
  Primeramente comprueba que la letra a contar existe en la palabra o frase, de no ser así, devuelve un *string* con "-1". Si está, recorre la palabra o la frase contando el número de veces que la letra aparece en ella.

  Finalmente devuelve el número de apariciones en un *string*.

1.2.22.2 Descripción de los atributos

- **palabra:** Palabra de donde contar la letra.

- **letra:** Letra a contar en la palabra.

1.2.23 LongitudPalabra

La clase LongitudPalabra es una subclase de la clase Funcion, encargada de gestionar la función que obtiene la longitud de una palabra.

Tiene como atributo un parámetro *string* privado llamado 'palabra' que representa la palabra la cual se va a medir.

1.2.23.1 Descripción de los métodos de la clase

- **LongitudPalabra**: Creadora de la clase a la cual le pasan un valor *string* por parámetro y lo almacena en el atributo privado de la clase.

- **execute**: Hace *Override* de la función abstracta de su padre, devuelve en un *string* la longitud de la palabra almacenada.

1.2.23.2 Descripción de los atributos

- **palabra:** Palabra de donde extraer su longitud.

1.2.24 ReemplazarPalabra

La clase ReemplazarPalabra es una subclase de la clase Funcion, encargada de gestionar la función que reemplaza un trozo de una palabra por otra dada.

Tiene cuatro atributos privados, dos son *strings*, representan las palabras a sustituir y la sustituida y dos enteros que indican las posiciones de la palabra original donde se quiere sustituir por el nuevo trozo.

1.2.24.1 Descripción de los métodos de la clase

- **ReemplazarPalabra**: Creadora de la clase a la cual le pasan un dos valores *string* por parámetro (palabra y trozo nuevo) y dos valores enteros (posiciones de la palabra) y los almacena en los atributos privados de la clase.

- **execute**: Hace *Override* de la función abstracta de su padre, devuelve en un *string* con "-1" si hay algún error en los atributos almacenados como: las posiciones no existen en la palabra, la palabra está vacía... Seguidamente se sustituye el trozo de la palabra original (almacenada) marcado por las dos posiciones almacenadas, por el nuevo trozo también almacenado. Finalmente se devuelve la nueva palabra

  Ej: Google-> "Google" , 4, 3, "d" -> Good

1.2.24.2 Descripción de los atributos

- **txt:** Texto del cual una parte va a ser sustituida.

- **pos:** Posición desde la cual comenzará la sustitución.

- **longd:** Número de carácteres a sustituir.

- **ntxt:** Texto que se cambiará por el antiguo.

1.2.25 ReemplazarCaracter

La clase ReemplazarCaracter es una subclase de la clase Funcion, encargada de gestionar la función que sustituye un carácter de una palabra por otro dado.

Tiene como atributo un parámetro *string* privado llamado 'txt' que representa la palabra original y dos más del mismo tipo que representan el carácter a sustituir y el nuevo..

1.2.25.1 Descripción de los métodos de la clase

- **ReemplazarCaracter**: Creadora de la clase a la cual le pasan tres valores *string* por parámetro y los almacena en los atributos privados de la clase.

- **execute**: Hace *Override* de la función abstracta de su padre, devuelve "-1" si el carácter a sustituir no existe en la palabra original. De lo contrario sustituye el carácter indicado por el nuevo.

  Ej: Hola-> "Hola", "o", "u" -> Hula

1.2.25.2 Descripción de los atributos

- **txt:** Texto del cual una parte va a ser sustituida.

- **cr:** Carácter que vamos a reemplazar.

- **nc:** Nuevo carácter a introducir.

1.2.26 Covarianza

La clase Covarianza es una subclase de la clase Funcion, encargada de gestionar la función covarianza.

Tiene como atributos dos vectores de *string* privados llamados 'valoresX' y 'valoresY', que representan los valores a utilizar en la covarianza.

1.2.26.1 Descripción de los métodos de la clase

- **Covarianza**: Creadora de la clase a la cual le pasan dos vectores *string* por parámetro y los almacena en los atributos privados de la clase.

- **execute**: Hace *Override* de la función abstracta de su padre, obtiene todos los valores que hay en los vectores almacenados y realiza la covarianza devolviendo el valor en un *string*.

1.2.26.2 Descripción de los atributos

- **valoresX:** Variables discretas X.

- **valoresY:** Variables discretas Y.

1.2.27 DesvEstandar

La clase DesvEstandar es una subclase de la clase Funcion, encargada de gestionar la función desviación estándar.

Tiene como atributos dos vectores de *string* privados llamados 'valoresX' y 'valoresY', que representan los valores a utilizar en la desviación estándar.

1.2.27.1 Descripción de los métodos de la clase

- **DesvEstandar**: Creadora de la clase a la cual le pasan dos vectores *string* por parámetro y los almacena en los atributos privados de la clase.

- **execute**: Hace *Override* de la función abstracta de su padre, obtiene todos los valores que hay en los vectores almacenados y realiza la desviación estándar devolviendo el valor en un *string*.

1.2.27.2 Descripción de los atributos

- **valores:** Valores necesarios de la función.

1.2.28 Media

La clase Media es una subclase de la clase Funcion, encargada de gestionar la función media.

Tiene como atributos un vector de *string* privado llamado 'valores', que representan los valores a utilizar en la media.

1.2.28.1 Descripción de los métodos de la clase

- **Media**: Creadora de la clase a la cual le pasan un vector *string* por parámetro y lo almacena en el atributo privado de la clase.

- **execute**: Hace *Override* de la función abstracta de su padre, obtiene todos los valores que hay en el vector almacenado y realiza la media de todos ellos devolviendo el valor en un *string*.

1.2.28.2 Descripción de los atributos

- **valores:** Valores necesarios de la función.

1.2.29 Mediana

La clase Mediana es una subclase de la clase Funcion, encargada de gestionar la función mediana.

Tiene como atributos un vector de *string* privado llamado 'valores', que representan los valores a utilizar en la media.

1.2.29.1 Descripción de los métodos de la clase

- **Mediana**: Creadora de la clase a la cual le pasan un vector *string* por parámetro y lo almacena en el atributo privado de la clase.

- **execute**: Hace *Override* de la función abstracta de su padre, obtiene todos los valores que hay en el vector almacenado y realiza la mediana de todos ellos devolviendo el valor en un *string*.

1.2.29.2 Descripción de los atributos

- **valores:** Valores necesarios de la función.

1.2.30 Varianza

La clase Varianza es una subclase de la clase Funcion, encargada de gestionar la función varianza.

Tiene como atributos un vector de *string* privado llamado 'valores', que representan los valores a utilizar en la varianza.

1.2.30.1 Descripción de los métodos de la clase

- **Varianza**: Creadora de la clase a la cual le pasan un vector *string* por parámetro y lo almacena en el atributo privado de la clase.

- **execute**: Hace *Override* de la función abstracta de su padre, obtiene todos los valores que hay en el vector almacenado y realiza la varianza de todos ellos devolviendo el valor en un *string*.

1.2.30.2 Descripción de los atributos

- **valores:** Valores necesarios de la función.

1.2.31 Pearson

La clase Pearson es una subclase de la clase Funcion, encargada de gestionar la función de coeficiente de Pearson

Tiene como atributos dos vectores de *string* privados llamados 'valoresX' y 'valoresY', que representan los valores a utilizar en la función.

1.2.31.1 Descripción de los métodos de la clase

- **Pearson**: Creadora de la clase a la cual le pasan dos vectores *string* por parámetro y los almacena en los atributos privados de la clase.
- **execute**: Hace *Override* de la función abstracta de su padre, obtiene todos los valores que hay en los vectores almacenados y realiza el coeficiente de Pearson devolviendo el valor en un *string*.

1.2.31.2 Descripción de los atributos

- **valoresX:** Variables aleatorias cuantitativas X.

- **valoresY:** Variables aleatorias cuantitativas Y.

1.2.32 Suma

La clase Suma es una subclase de la clase Funcion, encargada de gestionar la función que devuelve la suma de unos argumentos dados.

Tiene como atributo un parámetro *String[]* privado llamado 'valores' que representa los valores a sumar.

1.2.32.1 Descripción de los métodos de la clase

- **Suma:** Creadora de la clase a la cual le pasan un *String[]* y setea el atributo 'valores'.

- **execute**: Hace *Override* de la función abstracta de su padre y ejecuta la suma de los números que tiene almacenados en el atributo privado de la clase, devolviendo este valor como un string.

1.2.32.2 Descripción de los atributos

- **valores:** Valores a sumar.

1.2.33 Resta

La clase Resta es una subclase de la clase Funcion, encargada de gestionar la función que devuelve la resta de unos argumentos dados(arg0 - arg1 - ... - argN-1).

Tiene como atributo un parámetro *String[]* privado llamado 'valores' que representa los valores a restar.

1.2.33.1 Descripción de los métodos de la clase

- **Resta:** Creadora de la clase a la cual le pasan un *String[]* y setea el atributo 'valores'.

- **execute**: Hace *Override* de la función abstracta de su padre y ejecuta la resta de los números que tiene almacenados en el atributo privado de la clase, devolviendo este valor como un string.

1.2.33.2 Descripción de los atributos

- **valores:** Valores a restar.

1.2.34 Multiplicacion

La clase Multiplicacion es una subclase de la clase Funcion, encargada de gestionar la función que devuelve la multiplicacion de unos argumentos dados.

Tiene como atributo un parámetro *String[]* privado llamado 'valores' que representa los valores a multiplicar.

1.2.34.1 Descripción de los métodos de la clase

- **Multiplicacion:** Creadora de la clase a la cual le pasan un *String[]* y setea el atributo 'valores'.

- **execute**: Hace *Override* de la función abstracta de su padre y ejecuta la multiplicación de los números que tiene almacenados en el atributo privado de la clase, devolviendo este valor como un string.

1.2.34.2 Descripción de los atributos

- **valores:** Valores a multiplicar.

1.2.35 Division

La clase Division es una subclase de la clase Funcion, encargada de gestionar la función que devuelve la división de un conjunto de argumentos(arg0/arg1/.../argN-1).

Tiene como atributo un parámetro *String[]* privado llamado 'valores' que representa los valores a dividir.

1.2.35.1 Descripción de los métodos de la clase

- **Division:** Creadora de la clase a la cual le pasan un *String[]* y setea el atributo 'valores'.

- **execute**: Hace *Override* de la función abstracta de su padre y ejecuta la Division de los números que tiene almacenados en el atributo privado de la clase, devolviendo este valor como un string. Si alguno de los valores a excepción del primero es 0, devuelve como resultado un error.

1.2.35.2 Descripción de los atributos

- **valores:** Valores a dividir.

1.3 Explicación de los controladores del Dominio

1.3.1 ControladorDocumento

Esta clase es la clase encargada de procesar y realizar todas las operaciones y funciones que tengan que ver con un documento. Desde crearlo, añadir o eliminar hojas o incluso eliminarlo.

Esta clase tendrá tanto comunicación con el controlador de la capa de presentación para poder gestionar las peticiones del usuario que tengan que ver con documentos, como con el controlador ControladorDocumentoPersistencia de la capa de persistencia para poder almacenar y recuperar documentos.

1.3.1.1 Descripción de los métodos de la clase

- **cerrarDocumento:** Cierra el documento actual. Pone a null el documento referenciado.

- **crearDocumento:** Cierra el documento actual y crear un documento con una hoja con tantas filas y columnas como indican los dos enteros que le pasan como parámetros.

- **eliminaDocumento:** Esta función se encarga de ejecutar la funcionalidad de eliminar el documento actual, para ello llama a la función eliminaDocumento de la clase Documento y pone a null el atributo privado documento_ref.

- **añadirHoja**: añade una hoja al documento actual con el número de filas y columnas como indica los dos enteros que le pasan como parámetro.

- **eliminarHoja**: Elimina la hoja que tiene como identificador el entero que le pasan como parámetro del documento actual.

- **asignaNombreHoja:** Se le asigna un nuevo nombre a la hoja actual, este será el string que le pasan como parámetro.

- **renombrarDocumento:** Cambia el nombre del documento actual por el que le pasan como parámetro.
- **guardarDocumento:** Esta función se comunica con el ControladorDocumentoPersistencia y su función principal es la de ejecutar la funcionalidad de almacenar un documento. El path donde se almacenará es el string pasado como parámetro.

- **cargarDocumento:** Esta función se comunica con el ControladorDocumentoPersistencia y su función principal es la de ejecutar la funcionalidad de cargar un documento. El path donde se encuentra el documento a cargar es el string pasado como parámetro.

1.3.1.2 Descripción de los atributos

- **documento_ref:** Documento abierto actualmente.

1.3.2 ControladorHoja

El ControladorHoja se encarga principalmente de gestionar y realizar todas aquellas funciones y operaciones a nivel de una hoja. Tanto como añadir filas o columnas, como ejecutar algunas funciones que necesitan diversas celdas como lo son las funciones de estadística.

Esta clase tendrá comunicación con el controlador de la capa de presentación para poder gestionar las diferentes peticiones del usuario que tengan que ver con la hoja. Coge y procesa la información que necesite de las clases de dominio para que la capa de presentación pueda mostrar los cambios al usuario.

1.3.2.1 Descripción de los métodos de la clase

- **asignaHoja:** Se guarda en el atributo privado hojaAct la hoja identificada por el entero que le pasan como parámetro. (Esto es la hoja actual).

- **asignaCelda:** Se guarda en el atributo privado celdaRef la celda identificada por la posición identificada por los números que le pasan como parámetro. (Esto es la celda seleccionada actual).

- **renombrarHoja:** Asigna un nuevo nombre a la hoja actual, este es el que le pasan como parámetro.

- **addFilas**: Se añaden un conjunto de filas en blanco a partir de un número de fila. Las filas que habían en la posición pasada como parámetro y posteriores se moverán tantas posiciones hacia delante como indique el segundo parámetro que le pasan a la función.

- **addColumnas**: Se añaden un conjunto de columnas en blanco a partir de un número de columna. Las columnas que habían en la posición pasada como parámetro y posteriores se moverán tantas posiciones hacia delante como indique el segundo parámetro que le pasan a la función.

- **eliminarFilas:** Se elimina el conjunto de filas contiguas indicadas a partir de un número de fila pasado como primer parámetro. Las filas posteriores al conjunto de filas borrado re-ocuparan esas posiciones inferiores. El segundo parámetro indica el número de filas a eliminar.

- **eliminarColumnas**: Se elimina el conjunto de columnas contiguas indicadas a partir de un número de columna pasado como primer parámetro. Las columnas posteriores al conjunto de columnas borrado re-ocuparan esas posiciones inferiores. El segundo parámetro indica el número de columnas a eliminar.

- **ordenar:** Ordena ascendentemente o descendentemente las celdas del bloque seleccionado. El booleano pasado como parámetro servirá para saber de qué manera ordenarlo.

- **escribirContenido:** Esta función es la que se encarga de escribir el contenido y el valor en la celda seleccionada, dependiendo de si el parámetro que le pasan a esta función, es una función, una referencia o un simple texto, se escribirá un valor y un contenido en específico. Además si ocurriera un error, se escribiría un código de este en la celda.

- **esFechaValida**: Comprueba que la fecha introducida como parámetro sea correcta, devuelve true de serlo, false de lo contrario.

1.3.2.2 Descripción de los atributos

- **hojaAct:** Hoja que el usuario está visualizando actualmente.

- **celdaRef:** Celda que el usuario tiene seleccionada actualmente.

1.3.3 ControladorBloque

ControladorBloque es la clase que se encarga de gestionar algunas operaciones referentes a bloques de celdas, que son copiar un bloque, cortar y pegar un bloque. Para ello la clase necesitará de un bloque temporal copiado.

Esta clase tendrá comunicación con el controlador de la capa de presentación para poder recoger y utilizar la información necesaria de las clases de dominio y poder mostrarla correctamente.

1.3.3.1 Descripción de los métodos de la clase

- **copiar:** Crea un nuevo bloque temporal copiado y setea el valor del booleano cortar a falso.

- **cortar:** Crea un nuevo bloque temporal copiado y setea el valor del booleano cortar a true.

- **pegar:** Pega el contenido del bloque temporal copiado en la hoja actual y borra el contenido del mismo si el valor de cortar es true.

- **setBloqueSeleccionado:** Define los parámetros del bloque seleccionado.

- **clearBloqueSeleccionado:** Borra el bloque seleccionado.

1.3.3.2 Descripción de los atributos

- **bloqueCopiado:** BloqueTemporalCopiado activo actualmente. Cuando se sobreescribe, el anterior se deshecha y el *Garbage Collector* se encarga de liberar el espacio de memoria principal que ocupaba.

1.3.4 ControladorDominio

ControladorDominio es el controlador que se encargará de hacer de aislante entre capas, es decir, intercambiará llamadas con la capa de Presentación y con la capa de Persistencia. Siendo un controlador muy sencillo, contiene únicamente como atributos los otros tres controladores de dominio y un getter para cada uno de ellos.

1.3.4.1 Descripción de los métodos de la clase

- **getControladorDominio:** Getter del controlador, dado que es una clase singleton.

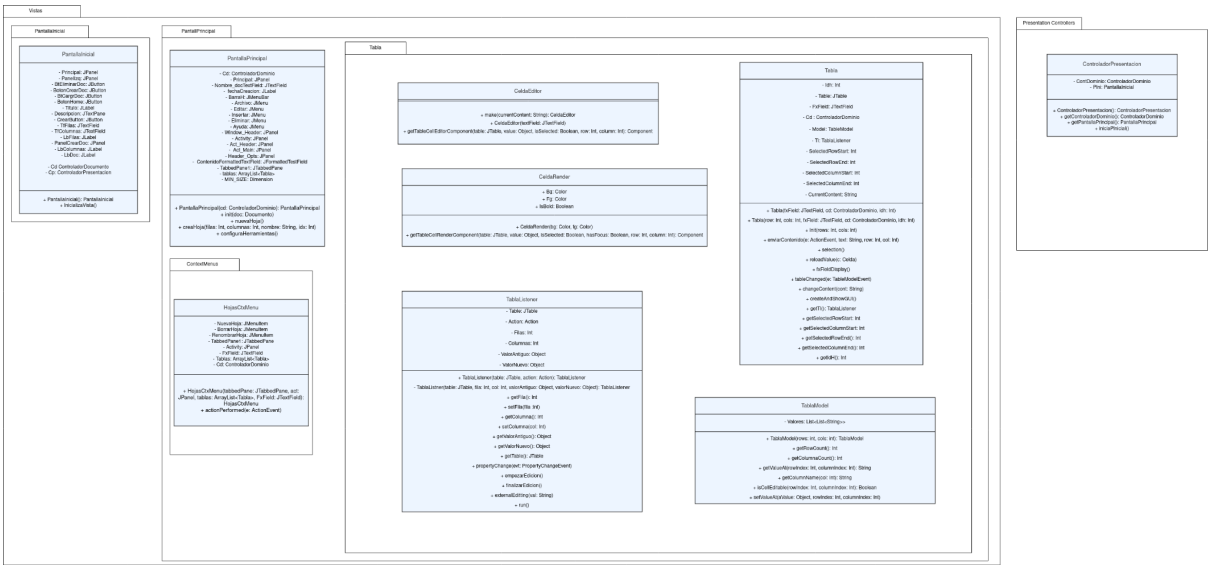
1.3.4.2 Descripción de los atributos

- **contBloque:** Única instancia del ControladorBloque.

- **contDocumento:** Única instancia del ControladorDocumento.

- **contHoja:** Única instancia del ControladorHoja.

2. Diagrama de clases y controladores de la capa de Presentación



2.0.1 MainProgram

Esta clase contiene el método principal para ejecutar el software, y concretamente se encarga de generar las vistas necesarias, así como hacerlas visibles.

2.1 Explicación de las vistas

2.1.1 Primera Vista

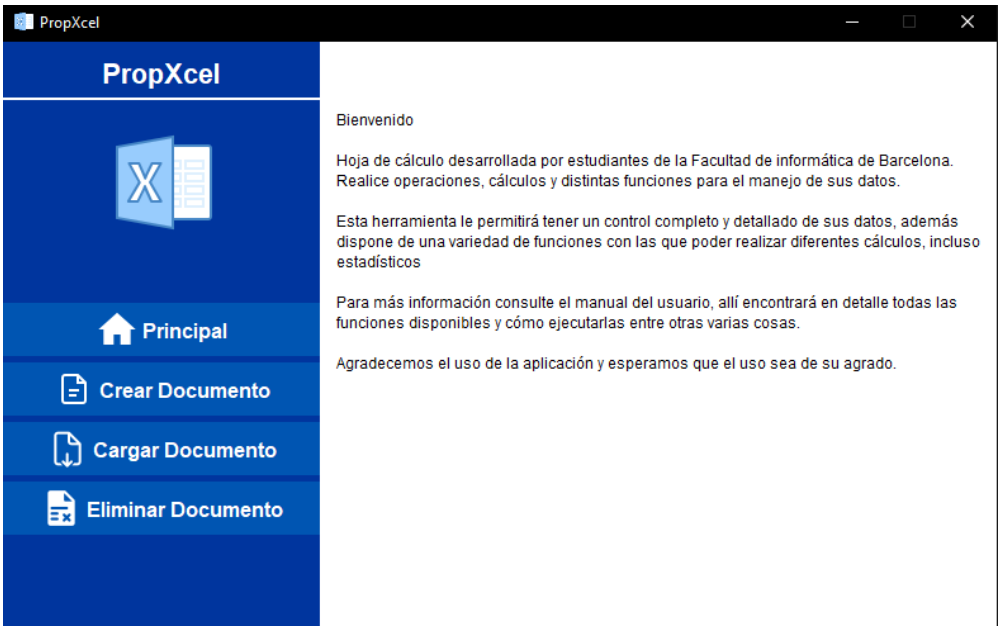
Esta vista será la pantalla inicial, es decir lo que verá el usuario nada más ejecutar la aplicación. Está implementada de tal manera que en vez de cambiar de vista cada vez que se presiona un botón, lo que cambiará son los distintos paneles que se van mostrando según el botón oprimido, por la tanto la vista sigue siendo la misma.

Esta vista está formada por dos *JPanel* principales, uno que engloba toda la vista y el segundo es el situado en la parte izquierda, de fondo azul oscuro, que contiene los cuatro botones, el título y la imagen de la aplicación.

En la parte derecha de la vista encontramos los componentes que irán cambiando según los botones que cliquemos. Encontramos dos de principales, un *JTextPane* que contiene una pequeña descripción de la aplicación y un *JPanel* que contiene dos *JTextField* para introducir el número de filas y columnas, y un botón para crear un documento con una hoja.

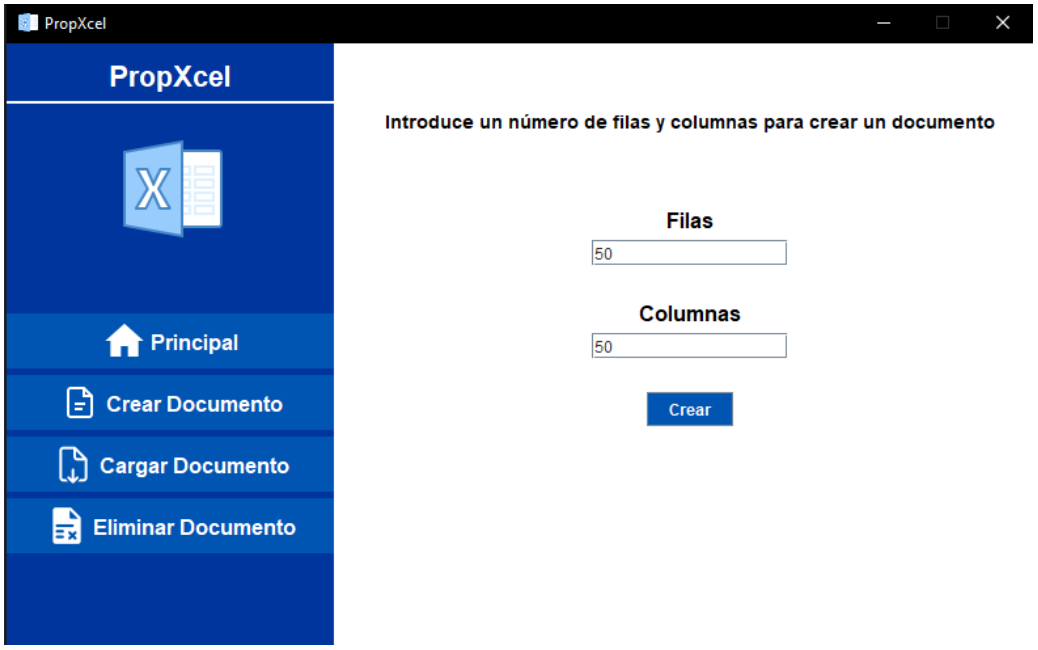
Tiene dos atributos privados ControladorDocumento y ControladorPresentacion que tienen que ver con las funcionalidades de esta. El Main program le pasará el controlador de presentación, después éste es el encargado de pasarles a las vistas los diferentes controladores que necesitan, por lo tanto le pasará a esta vista el controlador de documento que se lo guardará en el atributo privado para no tener que estar pidiendolo continuamente.

Principal



Por defecto se muestra este panel, aquí es donde se encuentra una breve introducción de la aplicación para informar al usuario. El botón “Principal” cuando es oprimido regresará a este panel.

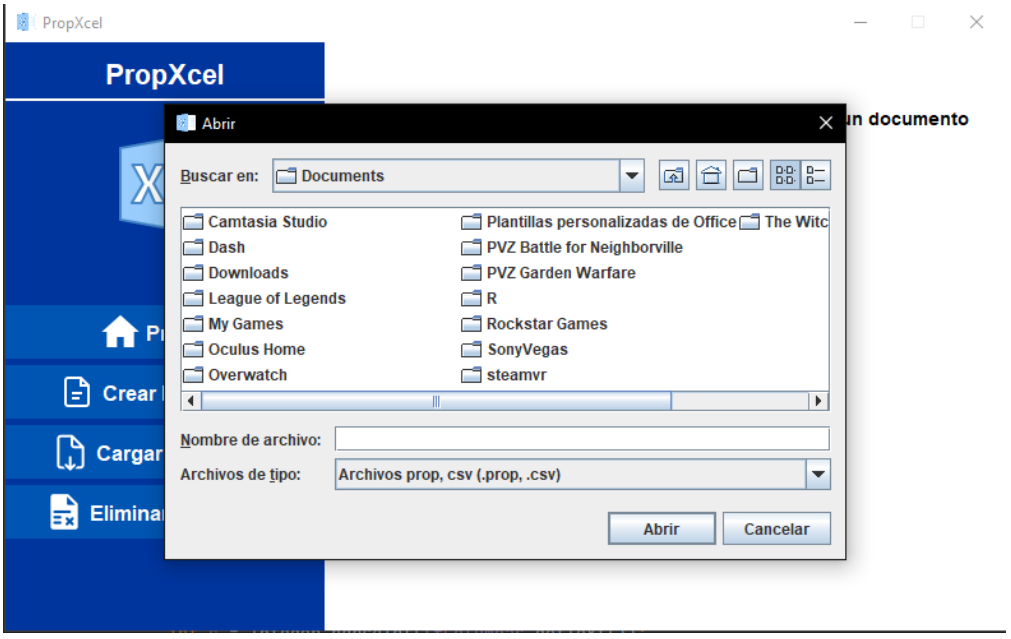
Crear Documento



Cuando se presione el botón “Crear Documento”, se cambiará a este panel. Como se ve en la imagen, el usuario tiene la opción de poder crear un documento con un número de filas y columnas que desee, de no introducir ningún número, por defecto se creará con 50 filas y 50 columnas como se muestra en la imagen.

Al presionar el botón “Crear” se creará el documento con los parámetros correspondientes y se cambiará a la segunda vista.

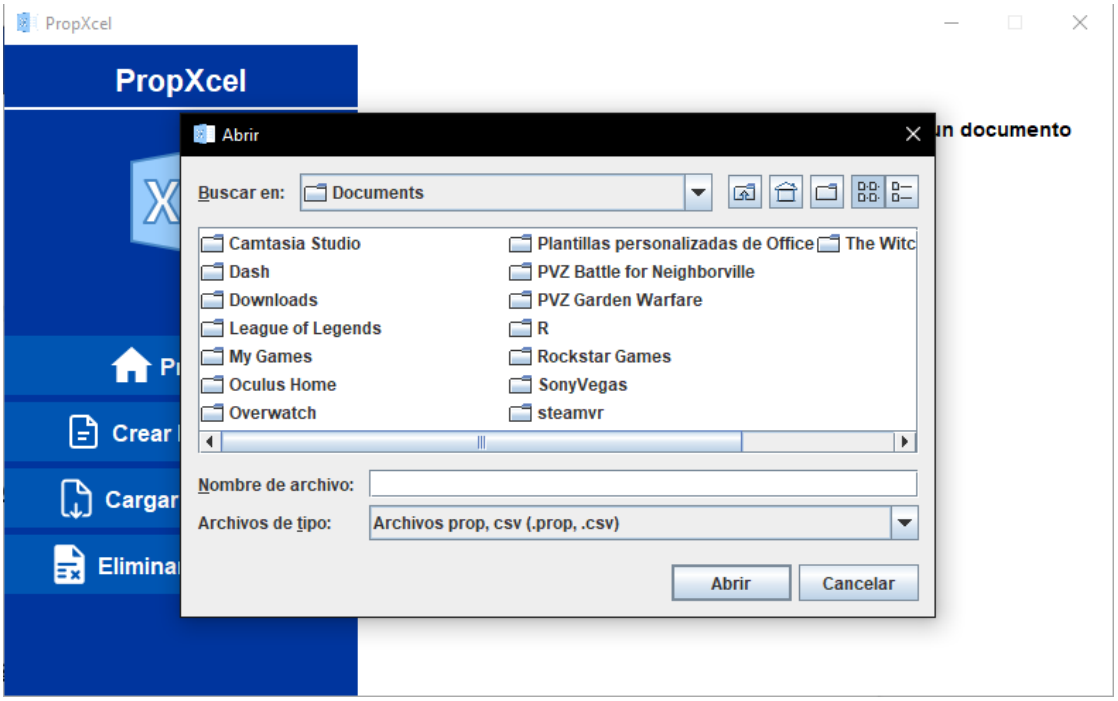
Cargar Documento



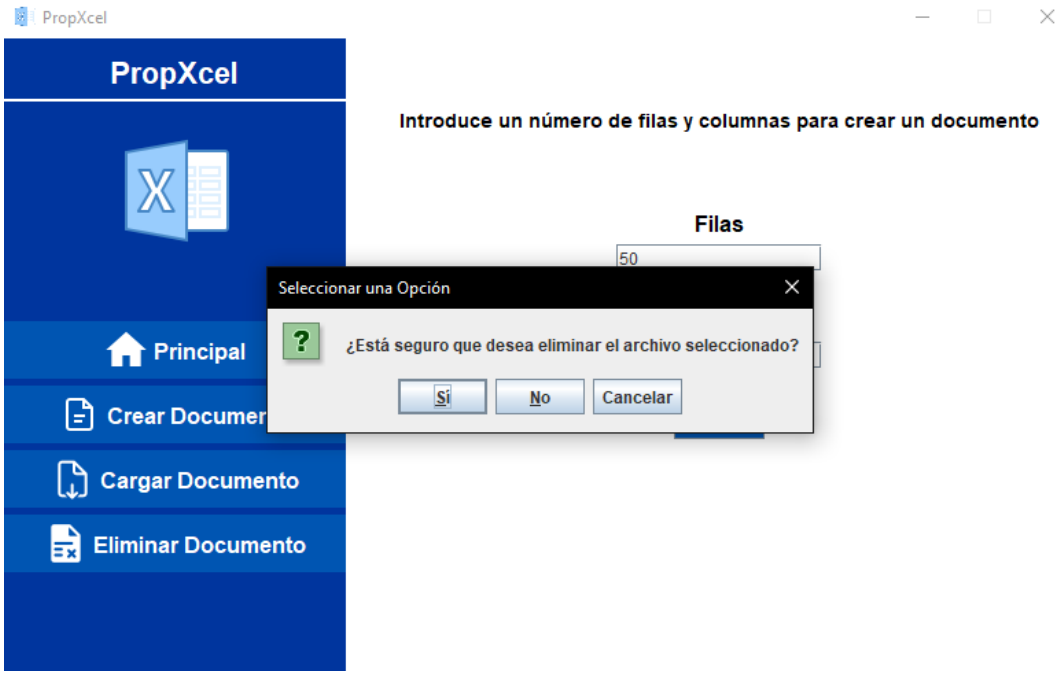
Cuando se presiona el botón “Cargar Documento” se abrirá una ventana de seleccionador de archivos del sistema con el filtro de archivos seleccionado para archivos .prop (archivos propios) y archivos .csv.

Una vez seleccionado el archivo y presionado “abrir” se abrirá el documento y se cambiará de vista a la segunda vista.

Eliminar Documento



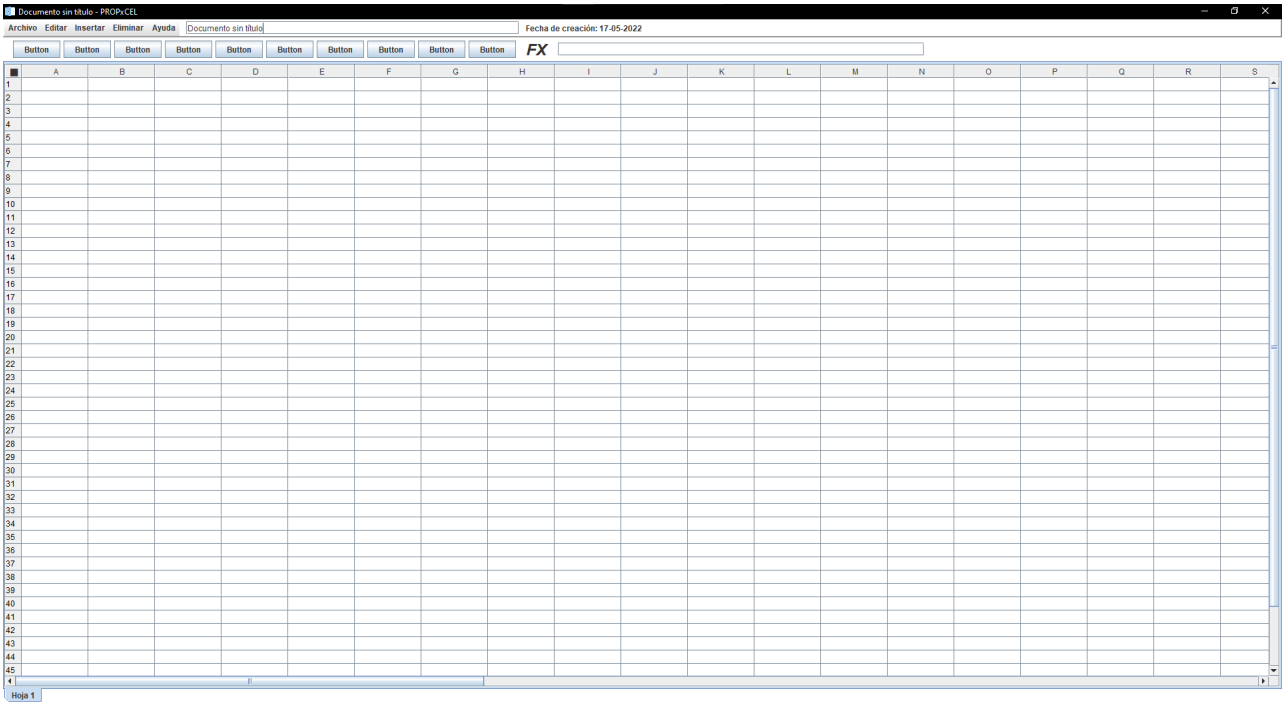
Cuando se presiona el botón “Eliminar Documento” se abrirá una ventana de seleccionador de archivos del sistema con el filtro de archivos seleccionado para archivos .prop (archivos propios) y archivos .csv, es decir solo se podrán seleccionar ese tipo de archivos.



Una vez seleccionado el archivo que el usuario desea borrar, el sistema lanzará una ventana preguntando al usuario si está seguro de eliminar el archivo que ha seleccionado, ofreciendo tres opciones.

Tanto como si pulsa no o cancelar, el archivo no se eliminará y se cerrará la ventana. Si el usuario presiona el botón de si, el sistema eliminará el archivo y cerrará la ventana de aviso.

2.1.2 Segunda Vista



Esta vista es la vista principal del programa, desde aquí el usuario podrá hacer uso de la mayoría de las funcionalidades principales de la hoja de cálculo.

Está formada por la barra de herramientas superior(Archivo, Editar, Insertar, Eliminar y Ayuda), un campo donde introducir el nombre del documento abierto, la fecha de creación del mismo, el campo de texto donde introducir las fórmulas aplicables a una celda, las pestañas con las Hojas existentes y por último, la tabla que contiene las celdas.

Barra de herramientas - Archivo

Este menú contiene las siguientes opciones:

- **Nuevo Documento:** Crea un documento en blanco.
- **Cargar Documento:** Carga un documento almacenado en disco.
- **Guardar Documento:** Guarda el documento actual en disco.
- **Nueva Hoja:** Crea una hoja nueva y ésta pasa a ser la visible.

- **Eliminar Hoja:** Si hay más de una hoja en el documento, elimina la hoja actual y la siguiente pasa a ser la visible. En caso contrario, salta un mensaje de error.

- **Renombrar Hoja:** Abre una ventana emergente donde introducir el nombre de la hoja visible actualmente.

- **Importar/Exportar:**

    - **Nueva Hoja(CSV):** Crea una nueva hoja a partir de un archivo CSV existente en disco.
    - **Importar Hoja(CSV):** Sobreescribe la hoja actual con el contenido de un archivo CSV existente en disco.
    - **Exportar Hoja(CSV):** Guarda el contenido de la hoja actual en un nuevo archivo CSV en disco.

- **Detalles:** Muestra los detalles del documento.

- **Salir:** Cierra la aplicación.

Barra de herramientas - Editar

Este menú contiene las siguientes opciones:

- **Cortar:** Corta el bloque seleccionado. Si no hay ninguno, no hace nada.

- **Copiar:** Copia el bloque seleccionado. Si no hay ninguno, no hace nada.

- **Pegar:** Pega el bloque almacenado. Si no hay ninguno, no hace nada.

- **Ordenar:** Ordena el bloque seleccionado actual de forma ascendente o descendente, según especifique el usuario. Si no hay ningún bloque seleccionado, ordena la hoja entera.

Barra de herramientas - Insertar

Este menú contiene las siguientes opciones:

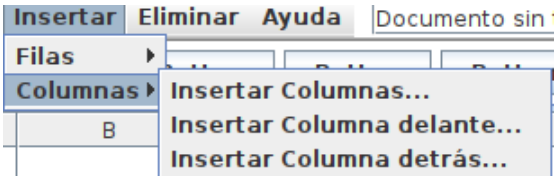
- **Filas:**

  - **Insertar Filas...:** Inserta n filas al final de la hoja.

  - **Insertar Fila delante...:** Inserta una fila posterior a la fila donde se encuentra la celda seleccionada.

  - **Insertar Fila detrás...:** Inserta una fila anterior a la fila donde se encuentra la celda seleccionada.

- **Columnas:**



  - **Insertar Columnas...:** Inserta n columnas al final de la hoja.

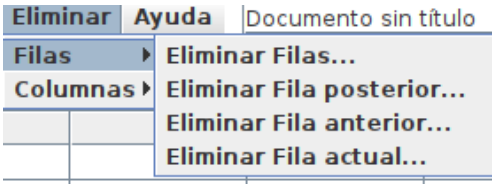
  - **Insertar Columna delante...:** Inserta una columna posterior a la columna donde se encuentra la celda seleccionada.

  - **Insertar Columna detrás...:** Inserta una columna anterior a la columna donde se encuentra la celda seleccionada.

Barra de herramientas - Eliminar

Este menú contiene las siguientes opciones:

- **Filas:**



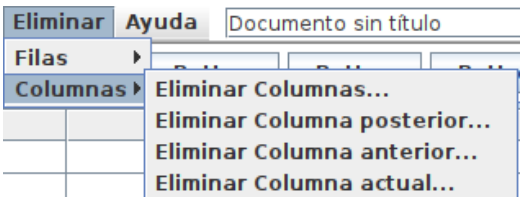
  - **Eliminar Filas...:** Elimina n filas del final de la hoja.

  - **Eliminar Fila posterior...:** Elimina la fila posterior a la fila donde se encuentra la celda seleccionada.

  - **Eliminar Fila anterior...:** Elimina la fila anterior a la fila donde se encuentra la celda seleccionada.

  - **Eliminar Fila actual...:** Elimina la fila actual donde se encuentra la celda seleccionada.

- **Columnas:**



    - **Eliminar Columnas...:** Elimina n columnas del final de la hoja.

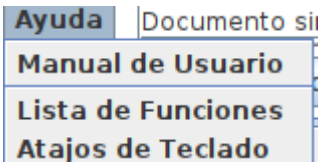
    - **Eliminar Columna posterior...:** Elimina la columna posterior a la columna donde se encuentra la celda seleccionada.

    - **Eliminar Columna anterior...:** Elimina la columna anterior a la columna donde se encuentra la celda seleccionada.

    - **Eliminar Columna actual...:** Elimina la columna actual donde se encuentra la celda seleccionada.

Barra de herramientas - Ayuda

Este menú contiene las siguientes opciones:



- **Manual de Usuario:** Abre el manual de usuario en una ventana emergente.

- **Lista de Funciones:** Abre una ventana emergente con una descripción de la lista de funciones.

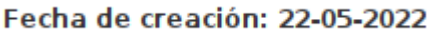
- **Atajos de Teclado:** Abre una ventana emergente con los atajos de teclado disponibles.

Campo del título del documento

El usuario podrá cambiar el nombre al documento, el cual inicialmente será “Documento sin título” y podrá ver reflejados los cambios en el propio cambio de texto así como en el título de la ventana.

Fecha de creación

Esta etiqueta contiene la fecha de creación del documento. Esta fecha no es actualizable y obtiene su valor en el momento de creación del documento.

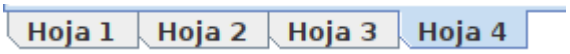


Campo de inserción de fórmulas

Este campo, denotado a su izquierda por el símbolo FX, será utilizado para hacer uso de las funciones, o para asignar el valor a una celda seleccionada en un momento dado.

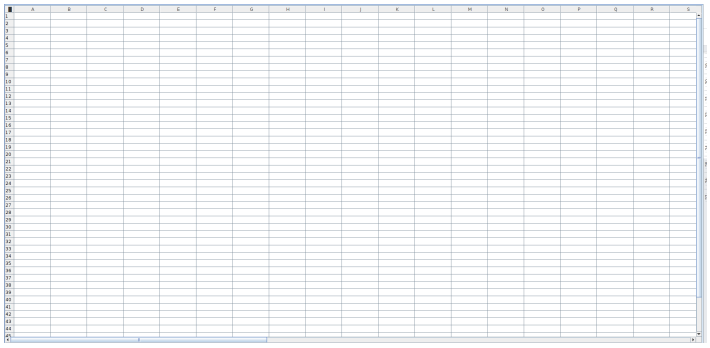
Pestañas de hojas agrupadas

Esta agrupación de pestañas tiene como finalidad que el usuario pueda intercambiar la hoja que está visualizando y/o editando en tiempo real. Cuando se le da click derecho a una pestaña, se abre un pequeño menú desde el cual podemos añadir una hoja, eliminar la hoja actual, o renombrar la hoja actual.



Tabla

Este componente es el principal de la vista. Contiene una matriz de celdas indexadas por una secuencia de valores alfabéticos en la parte superior(columnas) y una secuencia de valores numéricos en la parte izquierda(filas). Una celda se encuentra en una única hoja, y se halla en la intersección de las celdas de su fila y las de su columna.



2.1.2.1 Clases Auxiliares de la Vista

2.1.2.1.1 Tabla

La clase Tabla, es la clase que se utiliza en la vista principal para representar una tabla con filas y columnas de celdas donde poder escribir. Es hija de la clase de Java de JPanel e implementa la clase TableModelListener.

2.1.2.1.1.1 Descripción de los atributos de la clase

- **idh:** Un entero utilizado para identificar la hoja en la que está la tabla.
- **table:** Un componente JTable para poder posicionar las celdas en una hoja.
- **fxField**: Un JTextField para poder escribir valores y fórmulas.
- **cd :** ControladorDominio para poder ejecutar operaciones con celdas y hojas como escribir contenido en ellas.
- **model:** TableModel de la tabla de celdas.
- **tl:** TableListener para gestionar eventos en la tabla
- **selectedRowStart:** número de fila inicial necesario para definir una selección.
- **selectedRowEnd:** número de fila final necesario para definir una selección.
- **selectedColumnStart:** número de columna inicial necesario para definir una selección.
- **selectedColumnEnd:** número de columna final necesario para definir una selección.
- **currentContent:** String que almacena el contenido actual de una celda

2.1.2.1.1.2 Descripción de los métodos de la clase

- **Tabla:** Creadora de la clase que setea los valores que le pasan como parámetro (ControladorDominio, JTextField, idH, rows, colms).

- **init**: Inicializa la JTable con los valores de filas y columnas pasados como parametros.

- **enviarContenido:** Función utilizada para escribir un texto en una celda de la tabla.

- **changeContent**: Función para cambiar contenido dentro de la tabla por el string pasado como parámetro.

2.1.2.1.2 TablaListener

Esta clase implementa PropertyChangeListener y Runnable, y es la encargada de recoger los eventos que suceden en la tabla de la vista principal.

2.1.2.1.2.1 Descripción de los métodos de la clase

- **TablaListener(1):** Constructora a la que se le pasa por parámetro una JTable y un Action.

- **TablaListener(2):** Constructora a la que se le pasa por parámetro una JTable, un entero que corresponde a una fila, otro entero que corresponde a una columna, un valor antiguo y un valor nuevo.
- **propertyChange:** Recoge un evento y cambia la propiedad de la tabla a *editando*.

- **externalEditting:** Introduce un nuevo contenido a la tabla por un método externo a la propia tabla.

- **run:** Toma una acción en el thread en el que se está ejecutando.

2.1.2.1.2.2 Descripción de los atributos

- **table:** Tabla que contiene el listener.

- **action:** Acción llevada a cabo en un instante.

- **fila:** Fila sobre la cual aplicar la acción.

- **columna:** Columna sobre la cual aplicar la acción.

- **valorAntiguo:** Valor antiguo que tomaba la celda.

- **valorNuevo:** Valor nuevo que tomará la celda.

2.1.2.1.3 TablaModel

Implementa cuatro métodos de AbstractTableModel, aquellos que sirven para conseguir valores de la tabla.

2.1.2.1.3.1 Descripción de los métodos de la clase

- **TablaModel:** Dados el número de filas y el número de columnas de la tabla, setea el atributo privado de la clase con la cantidad de las mismas y un contenido vacío.

- **getColumnName:** Dado un número de columna, retorna el String correspondiente a esa columna.

- **isCellEditable:** Retorna cierto siempre.

- **setValueAt:** Dados un valor, una fila y una columna, setea el valor de la celda que corresponde a esos índices con el valor pasado por parámetro y actualiza la vista.

2.1.2.1.4 CeldaRender

Esta clase, la cual hereda de DefaultTableCellRenderer se encarga de los aspectos visuales de una celda: su background, su foreground, y si el contenido está en negrita.

2.1.2.1.4.1 Descripción de los métodos de la clase

- **CeldaRenderer:** Dados un background y un foreground de tipo Color cada uno, crea el objeto con la constructora de la clase padre y setea estos dos valores.

- **getTableCellRendererComponent:** Retorna la celda componente de la clase padre, con el background y el foreground previamente seteados.

2.1.2.1.4.2 Descripción de los atributos

- **bg:** Color de background de la celda.

- **fg:** Color de foreground de la celda.

- **isBold:** Booleano que vale true si el contenido de la celda debe mostrarse en negrita, falso en caso contrario.

2.1.2.1.5 CeldaEditor

La clase CeldaEditor es hija de la clase Java DefaultCellEditor. Es la clase que se encarga de asegurarse de que cuando editas una celda, se edita el contenido y no su valor.

2.1.2.1.5.1 Descripción de los métodos de la clase

CeldaEditor: Creadora de la clase que asigna el atributo textField heredado del padre con el textField pasado como parámetro.
make: Método que devuelve un CeldaEditor con un nuevo JTextField.

getTableCellEditorComponent: Se hace Override del método del padre, obteniendo un *Component* de los atributos pasados como parámetros, con el Contenido de la celda como texto a editar.

2.1.2.1.6 HojasCtxMenu

Esta clase hereda de JPopMenu e implementa sus ActionListeners. Es la clase que maneja el menú contextual de las Hojas en la parte inferior de la vista principal.

2.1.2.1.6.1 Descripción de los métodos de la clase

- **HojasCtxMenu:** Creadora de la clase. Dados un JTabbedPane, un JPanel, el ArrayList<Tabla> y un JTextField, setea sus atributos con estos e inicializa el menú pop up que se abre al hacer clic derecho sobre el context menu.

- **actionPerformed:** Dado un evento, lo captura y lo gestiona.

2.1.2.1.6.2 Descripción de los atributos

- **nuevaHoja:** Item dentro del menú pop up del context menu.

- **borrarHoja:** Item dentro del menú pop up del context menu.

- **renombrarHoja:** Item dentro del menú pop up del context menu.

- **tabbedPane1:** Pestañas que contienen las hojas.

- **Activity:** Panel que contiene el context menu.

- **fxField:** Campo para escribir la fórmula o valor de una celda.

- **tablas:** ArrayList que contiene las tablas de todas las hojas.

- **cd:** Controlador de dominio necesario para ejecutar los casos de uso correspondientes a el context menu.

2.2 Explicación de los controladores de Presentación

2.2.1 ControladorPresentacion

El propósito de este controlador es iniciar la visibilidad de la Pantalla Inicial descrita en el punto 2.1.1. Desde este controlador, las vistas podrán acceder al ControladorDominio para llamar a las funciones del resto de controladores y hacer uso de las funcionalidades que ofrece este sistema.

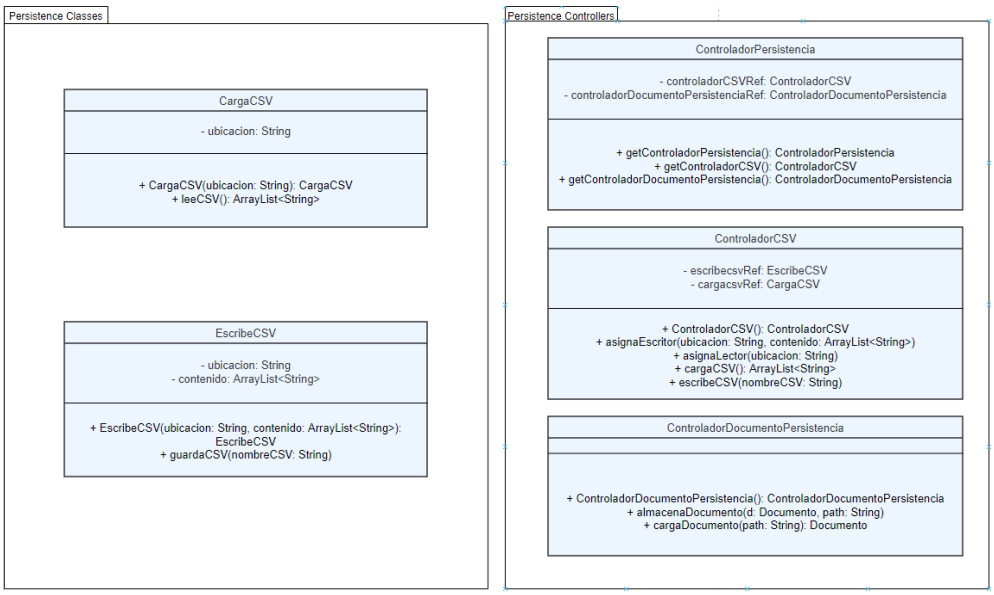
2.2.1.1 Descripción de los métodos de la clase

- **iniciaPInicial:** Setea el atributo pIni de la clase a una nueva PantallaInicial, y procede a inicializarla y posteriormente a hacerla visible.

2.2.1.2 Descripción de los atributos

- **pIni:** Atributo de tipo PantallaInicial. El controlador de la capa de presentación será el encargado de inicalizar esta vista y hacerla visible mediante el método iniciaPInicial().
-

3. Diagrama de clases y controladores de la capa de Persistencia



3.1 Explicación de las clases de Persistencia

3.1.1 CargaCSV

CargaCSV es la clase que se encarga únicamente de leer un archivo .csv de disco, procesar el contenido del mismo y preparar una estructura de datos para que las clases de dominio puedan recibirla(siempre mediante controladores como intermediarios) y exportar su contenido a una hoja que podrá ser nueva o sobreescrita con los valores del archivo leído.

Su creadora sólo requiere de la ubicación(*path* absoluto) del archivo a leer, la cual es especificada por el usuario.

3.1.1.1 Descripción de los métodos de la clase

- **CargaCSV:** Constructora que necesita de la ubicación del archivo a leer.

- **leeCSV:** Método principal de la clase, devuelve un ArrayList<String>, cuyos strings son líneas del archivo abierto en cuestión.

3.1.1.2 Descripción de los atributos

- **ubicacion:** Ubicación del archivo a leer, la misma debe ser absoluta y debe terminar en un archivo .csv. *Ej: /home/alumno/Documentos/input.csv*

3.1.2 EscribeCSV

Esta clase se encarga de lo opuesto a la anterior descrita: dada una estructura de datos(en nuestro caso para ambas clases, ArrayList<String>) que contiene el formato correcto para un archivo CSV, escribe cada una de las líneas en un archivo nuevo, cuyo nombre es especificado previamente por el usuario.

Tiene una única creadora, a la cual se le pasan por parámetro la ubicación del archivo a escribir(su directorio) y la estructura de datos que contiene el texto.

3.1.2.1 Descripción de los métodos de la clase

- **EscribeCSV:** Constructora que necesita de la ubicación del archivo a escribir y un ArrayList<String> con el contenido a escribir en el archivo.

- **guardaCSV:** Método principal de la clase, necesita del nombre del archivo CSV a guardar. Se encarga de escribir línea por línea el contenido del atributo “contenido”.

3.1.2.2 Descripción de los atributos

- **ubicacion:** Ubicación del archivo a leer, la misma debe ser absoluta y debe terminar en un directorio. *Ej: /home/alumno/Documentos/*

- **contenido:** ArrayList<String> cuyos strings son las líneas a escribir en el archivo.

3.2 Explicación de los controladores de Persistencia

3.2.1 ControladorCSV

El propósito de este controlador es dar de alta las instancias de las clases CargaCSV y EscribeCSV, así como llamar a sus métodos para la realización de los casos de uso relacionados con importación o exportación de CSVs. Sirve también como intermediario para generar el mínimo acoplamiento posible entre capas, aunque realmente los métodos que implementan estos casos de uso son muy sencillos.

Su creadora, en realidad, está vacía. Los métodos que dan de alta a las clases de la capa de Persistencia son también los que dan valor a los atributos de la clase. Cabe destacar que después de cada operación de lectura o escritura de CSV, la instancia de la clase utilizada para esa operación se borra, por lo que es necesario dar de alta otra(si así fuese necesario) mediante asignaLector() o asignaEscritor().

3.2.1.1 Descripción de los métodos de la clase

- **asignaEscritor:** Método que crea una nueva instancia de EscribeCSV y la asigna a escribecsvRef.

- **asignaLector:** Método que crea una nueva instancia de CargaCSV y la asigna a cargacsvRef.

- **cargaCSV:** Invoca la operación leeCSV() de cargacsvRef, y destruye el objeto a continuación.

- **escribeCSV:** Invoca la operación guardaCSV() de escribecsvRef, y destruye el objeto a continuación.

3.2.1.2 Descripción de los atributos

- **cargacsvRef:** Objeto de tipo CargaCSV. Necesario para efectuar las cargas de archivos csv.

- **escribecsvRef:** Objeto de tipo EscribeCSV. Necesario para guardar archivos csv.

3.2.2 ControladorDocumentoPersistencia

Este controlador implementa la serialización de un documento en su totalidad. Aseguramos que habrá únicamente una instancia en memoria ya que el ControladorPersistencia es singleton(ver apartado 3.2.3).

Sus métodos requieren de un *path* para guardar o cargar el documento en cuestión, y uno de ellos, almacenaDocumento(), también requiere el documento a guardar en cuestión.

3.2.2.1 Descripción de los métodos de la clase

- **almacenaDocumento:** Dado un Documento serializado y un *path* a un archivo, guarda ese documento en el archivo indicado.

- **cargaDocumento:** Dado un *path* a un archivo, lee el mismo y lo transforma a un Documento, el cual retorna.

3.2.3 ControladorPersistencia

Este controlador, el cual es singleton, guarda una instancia de el resto de controladores de la capa de Persistencia, y sirve como intermediario para efectuar operaciones que involucran a ésta capa y Dominio.

3.2.3.1 Descripción de los métodos de la clase

- **getControladorPersistencia:** Getter del ControladorPersistencia(singleton).

3.2.3.2 Descripción de los atributos

- **controladorCSVRef:** Instancia del ControladorCSV.

- **controladorDocumentoPersistenciaRef:** Instancia del ControladorDocumentoPersistencia.

4. Descripción de las estructuras de datos y algoritmos

4.1 Descripción de las estructuras de datos en java

HashMap: La clase HashMap utiliza hashing para implementar la interfaz map. Guarda los datos en pares de clave/valor y contiene una LinkedList de los nodos, llamado buckets, donde cada nodo es representado por una clase. Usa un array y una LinkedList para almacenar las claves y los valores. El tamaño predeterminado de HashMap es de 16 posiciones.

Para determinar en qué bucket se almacena un elemento con una clave determinada se calcula un índice mediante una función de hash sobre la clave y se crea un nuevo nodo que contiene su hash, la clave, el valor del nuevo elemento y un apuntador en siguiente nodo del mismo bucket.

La complejidad de las operaciones de HashMap en promedio son de O(1) pero en el peor de los casos pueden ser O(n).

ArrayList: La clase ArrayList está implementada como un array redimensionable. La estructura crece dinámicamente y se asegura de que siempre exista espacio libre. A la constructora puede especificarse la capacidad que tendrá, aunque si no lo especificamos, por defecto, la capacidad es de 10 elementos. Cada vez que se realiza una operación .add en la estructura se comprueba si hay espacio suficiente para albergar el elemento y en caso de no ser así aumenta en un 50% su capacidad.

Complejidad de las operaciones:

- .add(): Para añadir un solo elemento O(1). Para añadir n elementos O(n).
- .add(index,elemento): En media tarda O(n).
- .get(): tiempo constante O(1).
- .remove(): tiempo lineal O(n) ya que itera por la lista hasta encontrar el elemento.
- .indexOf(): en caso peor O(n).
- .contains(): está basado en la operación indexOf() por lo que tarda O(n).

Array: Un array en Java es una estructura de datos que nos permite almacenar una ristra de datos de un mismo tipo. El tamaño de los arrays se declara en un primer momento y no puede cambiar en tiempo de ejecución como puede ocurrir en otros lenguajes.

Complejidad de las operaciones:

- .length(): Obtiene el tamaño del array. Tarda tiempo constante O(1).
- Acceso aleatorio: Obtiene el elemento asociado a un índice. Como se mapea en memoria, se calcula la dirección y por tanto tarda tiempo constante O(1).

4.2 Descripción de las estructuras de datos por clases

4.2.1 Clase Documento

En la clase Documento utilizamos una tabla de hash (**HashMap**) para almacenar las hojas. Un documento no suele tener un número muy grande de hojas y por lo tanto la diferencia en eficiencia entre usar una estructura de datos u otra será negligible. Por este motivo, el criterio para elegir una tabla de hash es la simplicidad en la programación al añadir, eliminar y buscar elementos.

Utilizamos un HashMap de Integer - Hoja para almacenar las diferentes hojas que pueda llegar a tener un documento. Estas estarán almacenadas según un entero que corresponderá con su identificador.

4.2.2 Clase Hoja

Esta clase contiene una de las estructuras de datos más importantes del sistema, es la que se encarga de guardar todas las celdas en ella, por lo cual habrá que decidir una estructura de datos adecuada y concorde con el uso de las operaciones más comunes en celdas.

Analicemos las operaciones más y menos comunes. Por un lado, dado que el sistema se basa en una hoja de cálculo, la operación más común será seleccionar

una celda cualquiera para trabajar con ella, desde escribir un simple contenido a usarla en una función de otra celda. Esto implica que se necesitará un bajo coste en los accesos aleatorios. Esto podría indicar estructuras de datos como el *HashMap o Array de arrays*, ambas tienen un coste O(1) en selección de elementos aleatorios.

Otra operación algo común pero menos que la anterior, es la iteración por celdas en posiciones contiguas. En muchos casos, al ser una forma clara y organizada de estructurar la información en una hoja, los valores sobre los que se quiere realizar una función están en filas o columnas contiguas.

Esto llama a utilizar una estructura de datos como *LinkedList*, que por su implementación interna hace que el coste sea sensiblemente mejor para una iteración siguiendo un orden.

Finalmente las operaciones menos frecuentes usadas por los usuarios de hojas de cálculo suelen ser las inserciones de filas y columnas o la eliminación de estas.

Para ello una vez eliminada o insertada la fila o columna hará falta actualizar el resto de posiciones de las demás filas y columnas. Esto provoca un coste asintótico cuadrático en las estructuras comentadas para los otros casos.

Descrito todo lo anterior, nos hemos decantado por la estructura de datos *HashMap*.
Teniendo costes muy similares en las operaciones descritas con un *Array de arrays*, nos decantamos por esta estructura por la simplicidad a la hora de programar con ella. Además como añadido, en términos de espacio el *HashMap* es asintóticamente y en media mejor que el *Array de arrays*.

Por supuesto, la estructura restante que es *LinkedList de linkedlists*, queda descartada ya que el coste temporal de acceso a una celda es lineal, cosa que es peor que en las otras dos estructuras.

Cabe aclarar que la estructura será un *HashMap<Posicion,Celda>*, es decir un *HashMap* con una clave y valor de objeto propio. Si no hicieramos nada esto podria llevar a un error grave que afectaría a todo el funcionamiento, puesto que dos objetos *Posicion* serían diferentes, y al hacer un *.put()* de una nueva *Posicion*, la clave sería el objeto y no el par de enteros que definen a *Posicion*.

Por lo tanto, para que esto no pase, en la clase *Posicion*, redefinimos el código del ***hashCode*** y el método ***equals.*** Haciendo esto, permitimos tener un HashMap con clave propia y corregimos el posible error comentado antes.

Para la clase *Celda*, dado que no es la clave del *HashMap* no será necesario redefinir el código del ***hashCode***, pero para poder hacer comparaciones de valores y comprobaciones en el *HashMap*, tendremos que redefinir el método ***equals*** para asegurarnos que no se comete ningún error a la hora de realizar estas operaciones.

- Función transformaCSV

  · **ArrayList<String>** donde se guardan los distintos valores de las celdas en formato csv. Cada valor estará separado por ';' y casa fila por un salto de linea.

4.2.3 Clase Celda

En la clase Celda utilizaremos la estructura de datos LinkedList de Celdas para almacenar todas sus referenciantes. Dado que tendremos que iterar en algunas operaciones sobre todas ellas, e interaccionar con alguna de ellas LinkedList nos proporciona una ventaja en tiempo de recorrido respecto a sus contrincante el ArrayList, dado que el ArrayList es más rápido para el almacenamiento de datos, LinkedList lo es más para la manipulación de estos.

4.2.4 Clase BloqueTemporalCopiado

En la clase BloqueTemporalCopiado se utiliza un *Array de Arrays* de Celdas, este tiene la función de almacenar las celdas que temporalmente se han copiado, para desplazarlas o pegarlas en otras posiciones. Como vemos al no ser necesario el acceso a datos, simplemente se utiliza como almacén, y como conocemos el tamaño de antemano, lo más eficaz es utilizar la estructura de Arrays enfrente a otras, ya que dado a su tamaño fijo, esta es más rápida que otras, véase el ejemplo de su prima cercana ArrayList, al ser esta dinámica es algo más lenta.

4.2.5 Clase ObtenerMes

· ***Vector de String*** donde se guardan los nombres de los meses del año para asociarlos a su número extraído de la fecha.
· ***Vector de String*** auxiliar para guardar las diferentes partes de la fecha introducida para asociarlo con el vector de meses y poder mostrar el correspondiente.

4.2.6 Clase ObtenerAño

· ***Vector de String*** auxiliar para guardar las diferentes partes de la fecha introducida y poder mostrar la parte correspondiente.

4.2.7 Clase ObtenerDia

· ***Vector de String*** auxiliar para guardar las diferentes partes de la fecha introducida y poder mostrar la parte correspondiente.

4.2.8 Clase ObtenerNombreDia

· ***Vector de String*** donde se guardan los nombres de los días de la semana en castellano.
· ***Vector de String*** auxiliar para guardar las diferentes partes de la fecha introducida.
· ***HashMap de Integer – String*** para traducir los nombres de la semana en inglés al castellano. Se usa un *HashMap* por la facilidad de usar esta estructura como un diccionario.
· ***Vector de Month*** donde se guardan todos los nombres de los meses para poder usar las funcionalidades de LocalDate y asociarlo al número de la fecha introducida

4.2.9 Clase Traductor

- Función getArgumentosFuncion1aria

  · **ArrayList<Celda>** donde se guardan las distintas celdas de un intervalo para seguidamente poder obtener sus valores.

  · **Vector String** utilizado para guardar los valores pedido por una función

- Función getArgumentosFuncionNaria

  · **ArrayList<Celda>** donde se vuelve a utilizar para guardar las distintas celdas de un intervalo para seguidamente poder obtener sus valores.

  · **ArrayList<String>** utilizado para guardar los valores pedido por una función

Se usan *ArrayList* como estructuras por la facilidad de añadir elementos sin conocer el tamaño inicial y la eficiencia al recorrerlos siguiendo un orden.

4.2.10 Clase ControladorHoja

- Función ordenar

  · **ArrayList<Celda>** donde se guardan las distintas celdas de un bloque para seguidamente ordenarlas por sus valores.

- Función escribirContenido

  · ***Vector de String*** (*arg*, *arg*0, *arg*1, *arg*2, *arg*3, *arg*4) utilizados en algunos casos para guardar los diferentes argumentos que devuelve Traductor al invocar la función *getArgumentosFuncion1aria*, a utilizar en las funciones correspondientes.

· ***ArrayList<String[ ]>*** (argm) utilizado en algunos casos para almacenar los argumentos que devuelve *Traductor* al invocar la función *getArgumentosFuncionNaria* y poder gestionar esos valores y utilizarlos en la función correspondiente.
P.ej en la función que calcula la media de los valores.

· ***ArrayList<String>*** (argu) utilizado en algunos casos para extraer los valores de los argumentos que devuelve *Traductor* al invocar la función *getArgumentosFuncionNaria* y utilizarlos en la función correspondiente.

4.2.11 Clase Covarianza

· ***Vector de String***, utilizado en sus dos atributos privados para almacenar los diferentes valores con los que poder ejecutar la covarianza de estos.

4.2.12 Clase DesvEstandar

· ***Vector de String***, utilizado en su atributo privado para almacenar los diferentes valores con los que poder ejecutar la desviación estándar de estos.

4.2.13 Clase Media

· ***Vector de String***, utilizado en su atributo privado para almacenar los diferentes valores con los que poder ejecutar la media de estos.

4.2.14 Clase Mediana

· ***Vector de String***, utilizado en su atributo privado para almacenar los diferentes valores con los que poder ejecutar la mediana de estos.

4.2.15 Clase Pearson

· ***Vector de String***, utilizado en sus dos atributos privados para almacenar los diferentes valores con los que poder ejecutar el coeficiente de correlación de Pearson de estos.

4.2.16 Clase Varianza

· ***Vector de String***, utilizado en su atributo privado para almacenar los diferentes valores con los que poder ejecutar la varianza de estos.

4.2.17 Clase Suma

· ***Vector de String***, utilizado en su atributo privado para almacenar los diferentes valores con los que poder ejecutar la suma de estos.

4.2.18 Clase Resta

· ***Vector de String***, utilizado en su atributo privado para almacenar los diferentes valores con los que poder ejecutar la resta de estos.

4.2.19 Clase Division

· ***Vector de String***, utilizado en su atributo privado para almacenar los diferentes valores con los que poder ejecutar la división de estos.

4.2.20 Clase Multiplicacion

· ***Vector de String***, utilizado en su atributo privado para almacenar los diferentes valores con los que poder ejecutar la multiplicación de estos.

4.2.21 Clase CargaCSV

- Función eliminaComillas

  · ***Vector de String*** se usa para almacenar los datos que hay en el vector de string que le pasan como parámetro, pero sin comillas.

- Función leeCSV

  · ***ArrayList <String>*,** se usa para almacenar los datos que se van leyendo del documento en formato csv.

4.2.22 Clase EscribeCSV

· ***ArrayList <String>*,** atributo privado de la clase utilizado para almacenar los datos de una hoja en formato CSV.

4.2.23 Clase ControladorCSV

- Función cargaCSV

  · ***ArrayList <String>*,** utilizado para almacenar el ArrayList que contiene los datos del csv que devuelve la función *lee*CSV.

4.2.24 Vista PantallaPrincipal

· ***ArrayList <Tabla>*,** atributo privado de la vista utilizado para almacenar las diferentes JTable de las diferentes hojas.

4.3 Descripción de los algoritmos principales

Dado que este proyecto no tiene una fuerte carga de algoritmos, explicaremos los que consideramos más importantes para el mismo.

4.3.1 Asignar valor Celda

Este algoritmo se encuentra en la clase ControladorHoja, la función en concreto se llama escribirContenido. Cuando el usuario escribe un contenido en una celda, este tiene que ser entendido por nuestro sistema, de tal manera que sepa si está escribiendo una función o simplemente es un texto llano.

Para ello utilizaremos la función detecta de Traductor que nos devolverá qué tipo de contenido ha escrito el usuario y en caso de ser una función que tipo de función pide.

Mediante un switch ejecutaremos las diferentes posibilidades que puede haber escrito el usuario, dentro de cada caso se pedirán los atributos necesarios para la función a realizar a Traductor y se revisará que sean correctos, seguidamente se ejecutarán las funciones necesarias de cálculo de números, estadístico o de texto y se dejará el resultado del valor en la celda que se ha seleccionado, si por algún motivo ocurriera algún error, este deja en el valor de la celda un código de error en vez de un valor normal.

4.3.2 GetColumnaFila

Este algoritmo se encuentra en la clase Hoja, se encarga de devolver todas las celdas en filas o columnas definidas por una celda inicial y una celda final.

Dependiendo de la posición en que son pasadas la celda inicial y la celda final, tendremos 4 posibilidades de selección. En caso de fila, de derecha a izquierda o de izquierda a derecha y en caso de columna de arriba a abajo o de abajo a arriba. Esto lo sabremos mirando si la posición de la celda inicial es mayor o menor a la de la celda final.

Según lo explicado anteriormente se ejecutará un bucle (hay 4 distintos, 1 por caso), recogiendo las celdas de la estructura de datos comprendidas entre las posiciones pasadas como parámetro.

Dado que solo se puede cumplir una condición de los ifs internos a la vez y la función de los bucles internos de los ifs es la de recorrer las celdas desde la “celdaIncial” hasta la “celdaFinal” este algoritmo tendrá un coste de $\Theta(n)$.

4.3.3 GetArgumentosFuncion1aria

Este algoritmo es situado en la clase Traductor, recoge y devuelve los argumentos de una función 1aria a partir de un string con la función y sus parámetros.

Primero comprueba que el string que llega es una función, y que no se ha colado una string que pueda causar problemas al programa. Crea una lista, llamada "ret", donde se guardarán los argumentos necesarios. Si los argumentos contienen una referencia simple, añade el valor de esa referencia como parámetro. Si contienen una referencia compleja ($A1:$C1) , por cada celda en el rango, se añade su valor como parámetro. Si no contiene ninguna de estas, se añade directamente como valor a la lista de parámetros. Finalmente, devuelve la lista de argumentos.

Dado que el coste máximo de este algoritmo se da cuando se pasa una referencia del tipo fila o columna 'A1:A5', se usa la funcion getColumnaFila que tiene coste O(n), más luego el bucle que tendrá que recorrer todas ellas, el coste será de O(n + n) -> O(n).

4.3.4 GetArgumentosFuncionNaria

Situado en la clase traductor, el algoritmo recoge y devuelve los argumentos de una función Naria a partir de un string con la función y sus parámetros.

Crea una lista de arrays de Strings, y separa los parámetros que llegan. Por cada uno de estos, comprueba si es una referencia compleja ($A1:$C1) o simple, o si es un valor directo. Añade los argumentos correspondientes a la lista y la devuelve.

Este algoritmo se basa principalmente en recorrer todos lo argumentos que le pasan y a partir del tipo que sean obtener valor, lo que significa que de principio tenemos un coste de Θ (n).

Dentro del bucle principal, el coste máximo nos lo encontramos cuando se pasa una referencia del tipo fila o columna 'A1:A5', se usa la función getColumnaFila que tiene coste O(n), más luego el bucle que tendrá que recorrer todas ellas, el coste dentro del if será de O(n + n) -> O(n).

Por lo tanto el coste total del algoritmo será O(n*O(n)) -> O(n^2)

4.3.5 Añadir Filas / Columnas

Dado que aunque no sean 100% idénticos los algoritmos que añaden columnas o filas, al ser muy parecidos pero con pequeños matices distintos se describirán juntos.

Estos algoritmos se hallan en la clase ControladorHoja, sirven principalmente para añadir un conjunto de filas o columnas a partir de una fila o columna dada y un número.

Primeramente se moverán las filas o columnas que hay por encima de la posición indicada tantas veces como el número de filas o columnas se quieran añadir.
Esto se hace mediante un doble bucle for por lo que tendrá coste $\Theta(n^2)$.

Seguidamente se añaden las nuevas filas o columnas mediante otro doble bucle for por lo que también tendrá coste $\Theta(n^2)$.

Finalmente se actualiza el número de filas o columnas de la hoja.

Como se ve el coste total del algoritmo será: $\Theta(n^2 + n^2)$ -> $\Theta(n^2)$.

4.3.6 Eliminar Filas / Columnas

Dado que aunque no sean 100% idénticos los algoritmos que eliminan columnas o filas, al ser muy parecidos pero con pequeños matices distintos se describirán juntos. Cabe destacar que son similares a los de añadir filas o columnas pero su objetivo es el contrario.
Estos algoritmos se hallan en la clase ControladorHoja, sirven principalmente para eliminar un conjunto de filas o columnas a partir de una fila o columna dada y un número.

Primeramente se eliminan las filas o columnas contiguas indicado por el numero pasado a partir de la posición indicada. Esto se hace mediante un doble bucle for por lo que tendrá coste $\Theta(n^2)$.

Seguidamente se reasignan las posiciones a las filas o columnas restantes mediante otro doble bucle for por lo que también tendrá coste $\Theta(n^2)$.

Finalmente se actualiza el número de filas o columnas de la hoja. Como se ve el coste total del algoritmo será: $\Theta(n^2 + n^2)$ -> $\Theta(n^2)$.

4.3.7 Ordenar Bloques

Este algoritmo está situado en la clase ControladorHoja, se encarga de ordenar ascendentemente o descendentemente un bloque seleccionado.

Primeramente almacena todas las celdas del bloque seleccionado en una estructura de *ArrayList<Celda>*, esto se hace mediante un doble bucle for que tiene coste -> $\Theta(n^2)$.

Seguidamente se hace un *sort* de la estructura, ascendentemente o descendentemente según el booleano introducido. Esto tiene un coste en el peor caso de O(n log n) ya que *Collection.sort* de java implementa un *mergesort*.

Finalmente se vuelven a poner las celdas en el bloque seleccionado de manera ordenada mediante otro doble bucle for por lo cual su coste es -> $\Theta(n^2)$.
El coste total acaba siendo: $\Theta(n^2)$ + O(n log n) + $\Theta(n^2)$ -> $\Theta(n^2)$.