**Interfaces Graficas de Usuario**

**Aplicación para la representación gráfica de funciones en WPF**

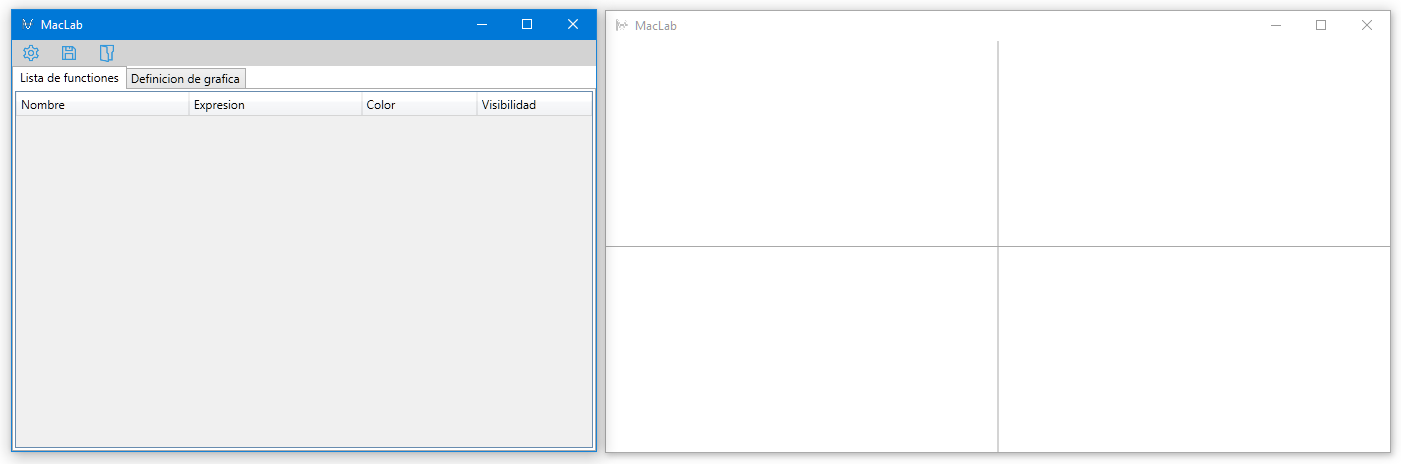
Francisco Pinto Santos

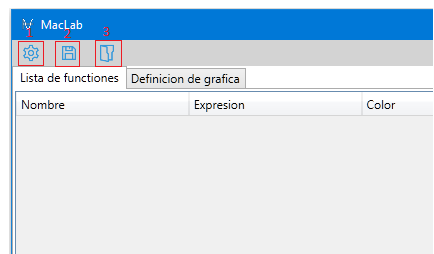
Curso 2018-19

**Manual de usuario**

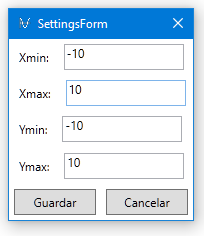
**Ventanas**

Al abrir el programa se desplegarán 2 ventanas:



La ventana mostrada a la izquierda en la figura (referida a partir de ahora como ventana de definición de representación) se utilizará para trabajar sobre la representación. Y en la ventana derecha (referida a partir de ahora para como ventana de representación) aparecerá la representación.

En la ventana de definición de representación en la parte superior se encuentran 3 botones.

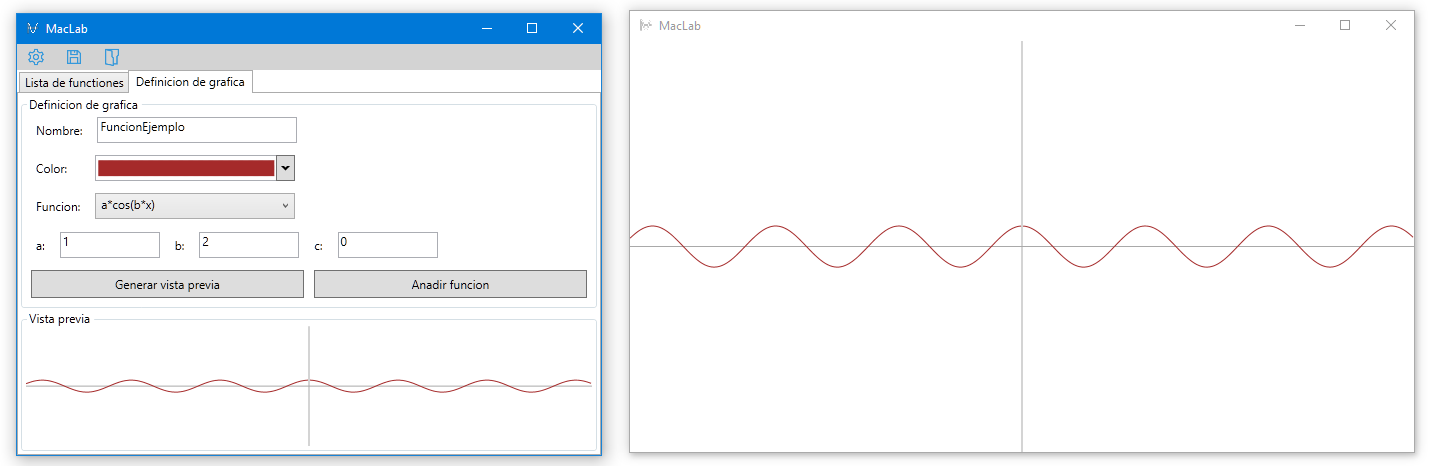
* El botón identificado como 1 en la imagen superior, se utilizará para desplegar el menú de configuración de representación (se explicará más adelante).
* El botón identificado como 2, se utilizará para guardar el proyecto actual en un fichero de extensión “.maclab”.
* Por último, el botón 3 se utilizará para importar un proyecto “.maclab”, lo cual ocasionará el borrado del progreso actual.

El Botón de configuración mencionado anteriormente desplegará la siguiente ventana, en la cual se podrá elegir el rango de valores de las variables X e Y, visibles en la ventana de representación.

**Añadir grafica**

En la ventana de definición de la representación hay dos pestañas: “Lista de funciones” y “Definición de gráfica”.

La pestaña “Definición de grafica”, presentada en la imagen inferior, será utilizada para añadir graficas a la representación:



En esta pestaña se presentan dos partes:

Un formulario llamado “Definición de grafica” donde se podrán definir los valores de la grafica a representar. Entre los cuales se encuentran Nombre, Color, función a representar y valores de las constantes implicadas en la función.

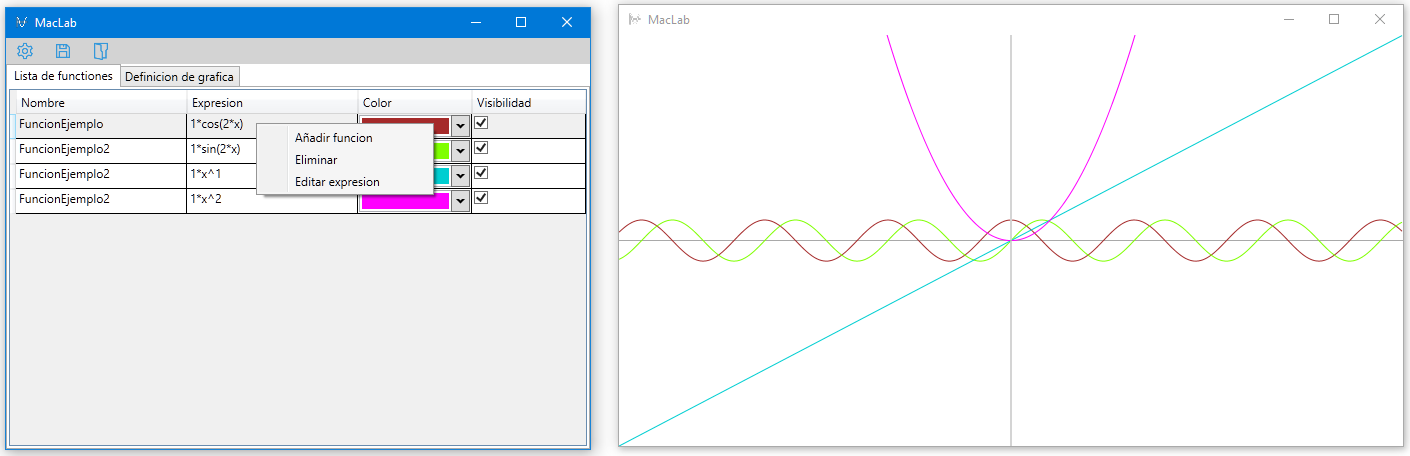
Y otra parte conocida como “Vista previa” en la cual se mostrará una vista previa de la función a añadir.

También se encuentran 2 botones:

* Generar vista previa: genera una vista previa de la función definida en el formulario superior en el panel de vista previa.
* Añadir función: añade la función a la ventana de representación.

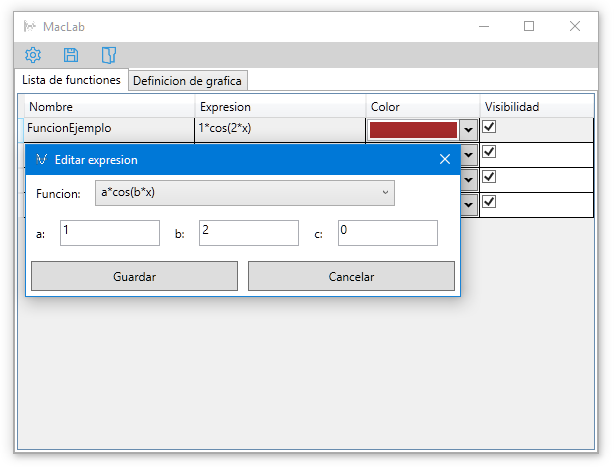
**Gestionar funciones:**

Para gestionar las funciones añadidas tenemos el panel “Lista de funciones”, en el cual como se ve en la imagen inferior se mantiene una lista de funciones activas, de las cuales se podrán modificar aspectos como el color, visibilidad y nombre.



En esta ventana se muestra un menú contextual en el cual , como se aprecia en la imagen superior:

* Añadir función: Desplaza el programa al pestaña “definición de grafica”.
* Eliminar: Si se realiza clic sobre una función, se eliminará de la representación y la lista de funciones activas.
* Editar expresión: Si se realiza clic sobre una función, nos desplegara la ventana que se muestra en la imagen inferior, con la cual se podrán editar los aspectos relacionados con la expresión que define la grafica correspondiente a esa función.

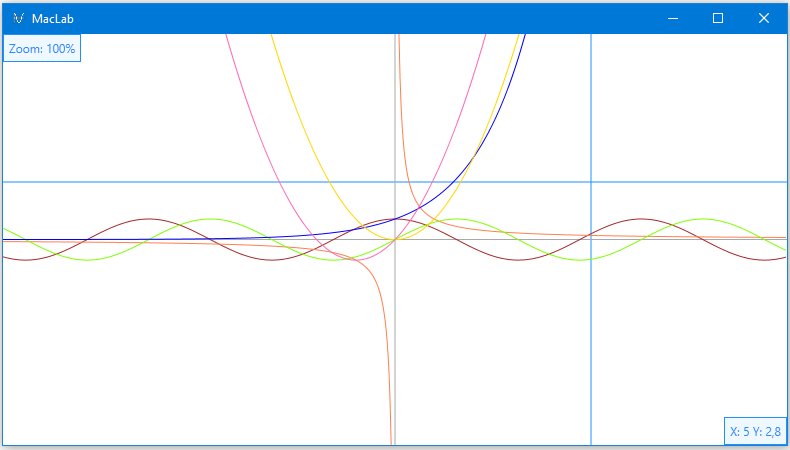


**Trabajando con la representación:**

En el panel de representación según se pasa el cursor por encima, en la esquina inferior izquierda aparece una etiqueta con la posición en la que se encuentra el cursor y unos ejes centrados en la posición de este.

También hay un menú contextual en esta ventana, con la única opción de exportar imagen; con la cual se nos permitirá exportar la representación actual como una imagen png.

Por último cabe destacar, que podremos hacer zoom mediante la ruleta del ratón, permitiéndonos ir desde un 10% hasta un 500%;

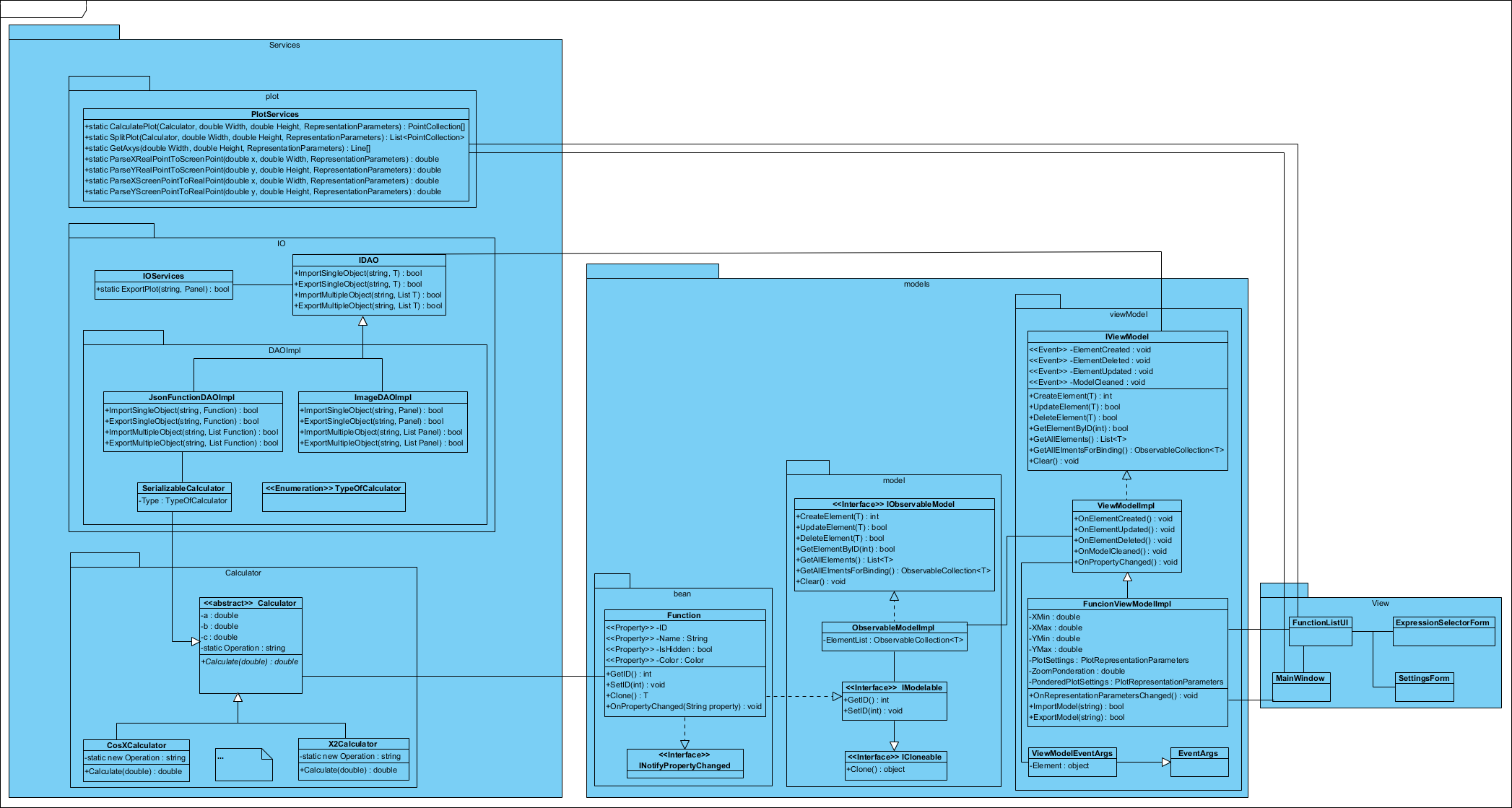


**Manual de programador**

Se explicará la estructura y funcionamiento del programa dividido por paquetes.

Cabe destacar que la aplicación esta basada en un VMMV, en el cual la lógica de negocio se ha encapsulado en el paquete “services”.

La razón principal de uso del patrón VMMV es porque facilita trabajar con WPF, y sincronizar varias ventanas.



**Modelo**

Se encuentra en el paquete “models”.

Aquí se encuentra la funcionalidad central de la aplicación, y está dividida en tres subpaquetes:

* Model: aquí se encuentran a su vez 3 elementos:
  + IObservableModel: Es una interfaz con tipos parametrizados que define un modelo basado en gestión de IDs, y provee las operaciones básicas de CRUD, además de la operaciones de “Clear“ y “GetAllElements“. La peculiaridad que tiene es que trabaja con una “ObservableCollection“, por eso ofrece un método “GetAllElementsForBinding“, lo que hará posible el que se saquen los elementos que tiene almacenado el modelo para poder hacer un enlace con cualquier control.
  + IObservableModelImpl: Es una clase que implementa la interfaz “IObservableModel“ para tipos parametrizados también.
  + IModelable: Interfaz que tiene que implementar todo elemento que quiera ser almacenado en un “IObservableModel“, que tiene 3 funciones básicas para tratamiento de objetos con IDs que son GetID(), SetID() y Clone() (este último heredado de ICloneable).
* ViewModel: en este paquete se implementa el modelo de la vista, para lo cual se han utilizado 3 elementos:
  + IViewModel: Es una interfaz con tipos parametrizados que provee las mismas operaciones que “IObservableModel“, pero a su vez tiene una serie de eventos básicos que son lanzados cuando un elemento es creado, eliminado o modificado.

Además, en este fuente están definidos el delegado “ViewModelEventHandler“ (que es el usado para definir los eventos recién mencionados), y la clase “ViewModelEventArgs“, la cual es usada como parámetro de argumentos para los eventos del modelo, y contiene el elemento afectado por la acción que ha disparado el evento.

Por último, sobre esta interfaz cabe destacar que se sustenta sobre un “IObservableModel“, por eso cualquier elemento que se quiera introducir en este modelo debe implementar “IModelabe“.

Además, los objetos que entren en este modelo deben implementar la interfaz “INotifyPropertyChanged“ porque si van a ser utilizados para hacer un enlace con la vista, cuando se cambie una propiedad, buscamos que se provoque un evento que haga al modelo de la vista lanzar otro evento del tipo “ElementUpdated“.

* + IViewModelImpl: Implementa la interfaz “IViewModel“ con tipos parametrizados y se sustenta sobre un “IObservableModelImpl“.
  + FunctionViewModelImpl: Es un “IViewModelImpl“ para funciones, además de que añade la gestión propiedades de representación de la gráfica ( estas están almacenadas en un “struct RepresentationParameters“, definido en el mismo fuente), y el factor de ponderación de zoom, el cual se utiliza para ponderar los campos del struct recién mencionado para provocar una sensación de zoom en el usuario.

Añade un evento “RepresentationParametersChanged” que se dispara cuando algún parámetro de representación cambia

* Bean: Aquí se encuentran los objetos planos, en este caso, únicamente la clase “Function“ que es el objeto utilizado para guardar la información de las funciones.
  + Tiene 5 propiedades que son ID, Name, Color, Calculator y IsVissible.
  + Están presentes en el varias etiquetas “ [JsonIgnore] “ y “ [JsonConstructor] “ que son para indicarle a la biblioteca Json.Net como serializar y deserializar el objeto.
  + Implementa la interfaz “IModelable“ de la que hemos hablado antes por lo que presenta los métodos “GetID()“, “SetID()“ y “Clone()“.
  + Implementa la interfaz “INotifyPropertyChanged“, luego por cada propiedad que se desea enlazar ( todas menos ID), se tiene otra propiedad llamada “NombrePropiedadProperty”. Además, tiene la función “OnPropertyChanged “ para cumplir con las normas de estilo de los eventos.

**Funcionalidad**

Se encuentra en el paquete “services”.

Aquí están presentes 3 subpaquetes:

* calculator: en este paquete se encuentra implementado un mecanismo que elabore para poder hacer un tratamiento genérico del cálculo de las funciones.

Por una parte, está la clase Calculator, que es una clase abstracta, con tres propiedades que son a, b y c que son double, las cuales serán los parámetros usados a la hora de definir las funciones (ej: a\*cos(b\*x)).

Tiene una propiedad estática adicional, que es un string llamado Operation, la cual será redefinida en cada implementación e indicará que operación realiza dicha implementación.

Esta clase tiene 2 métodos abstractos que son Calculate y Clone.

Calculate será redefinido en cada clase que lo herede, definiendo una operación para calcular el valor Y de una función para un valor X determinado.

Entre los Calculate implementados se encuentran CosXCalculator, SinXCalculator, XExpNCalculator, …

También he implementado un Calculator adicional llamado MultipleOperationCalculator, de tal forma que en el futuro si se desean implementar Calculator que realicen operaciones complejas, solo se le debe dar un array de Calculator para que realicen dichas operaciones en orden secuencial; pudiéndole dar como valores de ese array otros MultipleOperationCalculator, creando así un árbol de operaciones lo cual nos posibilita hacer cualquier operación.

* IO: Aquí se encuentran implementados unos mecanismos de entrada salida muy primitivos.

Por una parte, tenemos una interfaz DAO (la cual tiene 4 métodos muy genéricos para exportar e importar un único objeto, o una lista de ellos) y sus implementaciones.

Dicha interfaz tiene 2 implementaciones:

* + ImageDaoImpl: en esta implementación se realiza la transformación de un panel cualquiera a imagen, y se exporta.
  + JsonDaoImpl: aquí se implementan los mecanismos necesarios para realizar la entrada y salida de datos del modelo en forma de JSON, para poder trabajar con ficheros de proyecto .maclab, los cuales son una lista de funciones serializadas a JSON.

Este fichero se basa totalmente sobre la biblioteca Json.net.

Para poder realizar la serialización y deserialización de los objetos Function aquí se encuentra implementado llamado un Calculator SerializableCalculator, el cual sirve para poder guardar los datos normales de un Calculator, y además de que tipo es (CosXCalculator, SenXCalculator, …) mediante un atributo adicional que es una enumeración.

Todas las funciones serializadas, lo están mediante un SerializableCalculator, el cual se convierte en un Calculator normal, una vez que se han deserializado todas las funciones.

Por otra parte, tenemos una clase llamada IOSerivces que es una clase con métodos estáticos, la cual sirve para hacer tareas de IO que no se centran en los datos del modelo, como puede ser exportar como imagen el panel de las funciones.

* Plot: contiene “PlotSerivces”, que es una clase que reúne una serie de funciones estáticas, las cuales se utilizan para realizar tareas como generar ejes de coordenadas, generar los puntos para representar una función, separarla en segmentos, … y para realizar cálculos de transformación de puntos de pantalla a reales y viceversa.

**Utilidades**

Se encuentra en el paquete “utils”.

Contiene una única clase, llamada “Constants” que tiene las constantes utilizadas en la aplicación.

**Vista**

Se encuentra en el paquete “view”.

Esta compuesta de 4 ventanas:

* ExpressionSelectorForm: es un formulario bastante simple que nos permite editar todos los detalles de la expresión de una función.

Si se pulsa el botón de guardar, se comprueba si los datos son correctos.

Cabe destacar que tiene un evento mostrar y ocultar los campos correspondientes al valor “C”, cuando este no es necesario.

* FunctionListUI: Esta es una ventana secundaria no modal, en la que tendremos dos pestañas:
  + La primera consiste en un datagrid bindeado a el modelo, de tal forma que se pueden eliminar y editar las gráficas con repercusión directa en la representación ya que la clase Function implementa la interfaz INotifyPropertyChanged.

En lo relativo a los eventos de esta pestaña están registrados aquellos que nos permiten realizar las acciones de edición y eliminación sobre las funciones, en las cuales se toma el elemento seleccionado del datagrid y se realiza la acción pertinente en el modelo.

Cabe destacar que, para editar la expresión de las funciones, se usa una ventana del tipo ExpressionSelectorForm.

* + La segunda pestaña nos permite añadir funciones y generar una vista previa en el caso de que deseemos.

Para realizar la vista previa se realizan los mismos mecanismos que en la representación de funciones normal.

Esta ventana también presenta una toolbar con los botones de exportar e importar proyecto, para lo cual se realiza una serialización o deserialización respectivamente, de la lista de objetos Function del modelo, a formato json.

En dicha toolbar también esta presenta un botón de ajustes, que nos desplegara una ventana SettingsForm.

En esta ventana el tratamiento de datos se realiza en de la misma manera en todos los sitios: se tomarán los datos si el diálogo a acabado correctamente y los guardara en el modelo, en el caso que se desee añadir. O se toma la función seleccionada en el datagrid y se realizan las acciones pertinentes sobre el modelo.

* SettingsForm: Es una ventana que muestra un formulario para establecer los límites de representación de las graficas en los ejes de abscisas y ordenadas.

En cuanto al contenido es bastante simple, ya que solo presenta 4 campos y dos botones: cancelar y guardar, y si se pulsa guardar se comprueba que los datos introducidos son correctos.

* MainWindow: Es la ventana principal, en la cual se muestra la representación gráfica.

Aquí entre otras cosas están los eventos del modelo, los cuales son disparados cuando las funciones se crean, eliminan, …

Cabe destacar que las representación de una función es un array de Polilyine.

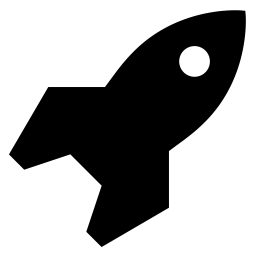
A parte de lo dicho también están registrados los eventos necesarios, para hacer zoom, mostrar la posición en la que se encuentra el cursor, refrescar el panel cuando cambian las condiciones de representación, exportar imagen, …

**Referencias y bibliotecas usadas**

**Bibliotecas usadas**

He usado 2 bibliotecas para realizar este trabajo. Ambas instaladas con el gestor de paquetes NuGet de Visual Studio.

Json.Net

La cual ha sido usada para serializar y deserializar con JSON los ficheros .maclab, ya que estos son un array de las funciones creadas por el usuario serializado a JSON.

https://www.newtonsoft.com/json

Wpf extended toolkit

Esta biblioteca ha sido usada para tomar el control “ColorPicker”, el cual ha se ha utilizado para usar seleccionar el color de las funciones en la tabla de funciones y en el la parte de definición de función.

https://github.com/xceedsoftware/wpftoolkit

**Recursos bibliográficos**

Libros consultados

Apuntes de la asignatura Interfaces Graficas de usuario.

Webs visitadas

[csharphelper.com](http://csharphelper.com) – he utilizado como ejemplo una serie de publicaciones que tienen sobre como dibujar graficas en C#

[docs.microsoft.com](https://docs.microsoft.com) – documentación sobre WPF, C# y Visual Studio

[icons8.com](https://icons8.com) – de aquí tome los iconos que necesite para la aplicación

[wpf-tutorial.com](https://wpf-tutorial.com) – lo consulte para aprender a usar controles mas avanzados como el datagrid, y para consultar como usar el dispatcher.

<https://stopbyte.com> – varias dudas de carácter general sobre WPF, entre ellas:

* <https://stopbyte.com/t/how-to-catch-datagrid-cell-content-value-changed-event-with-sourceupdated-on-updatesourcetrigger/61>

[stackoverflow.com](https://stackoverflow.com/) – varias páginas para resolver desde dudas básicas sobre C# hasta como utilizar controles complejos. No conservo todas, pero algunas de ellas son:

* <https://stackoverflow.com/questions/6462226/wpf-datagrid-binding-doesnt-update-until-clicking-row-header>
* <https://stackoverflow.com/questions/3913580/get-selected-row-item-in-datagrid-wpf>
* <https://stackoverflow.com/questions/6320885/add-a-new-row-to-datagrid-at-runtime-wpf>
* <https://stackoverflow.com/questions/4983951/how-do-i-bind-a-listcustomobject-to-a-wpf-datagrid>
* <https://stackoverflow.com/questions/43793630/property-names-are-not-available-for-my-datagrid-column-bindings>
* <https://stackoverflow.com/questions/2032832/how-to-reference-right-clicked-object-in-wpf-context-menu-item-click-event-handl>
* <https://stackoverflow.com/questions/47004607/xctk-colorpicker-in-datagrid-wpf>
* <https://stackoverflow.com/questions/13374270/dynamic-data-display-wpf-need-to-add-text-to-canvas-c-sharp>
* <https://stackoverflow.com/questions/11878217/add-items-to-combobox-in-wpf>
* <https://stackoverflow.com/questions/13680477/wpf-window-sizechanged-event-after-binding>
* <https://stackoverflow.com/questions/4430090/how-to-draw-a-function-plot-in-wpf>
* <https://stackoverflow.com/questions/1132967/scrollviewer-not-scrolling-in-wpf>
* <https://stackoverflow.com/questions/8881865/saving-a-wpf-canvas-as-an-image>
* <https://stackoverflow.com/questions/5101895/how-to-change-title-bar-image-in-wpf-window>
* <https://stackoverflow.com/questions/17515631/add-an-image-in-a-wpf-button>
* <https://stackoverflow.com/questions/11133947/how-do-i-open-a-second-window-from-the-first-window-in-wpf>
* <https://stackoverflow.com/questions/3970522/wpf-add-a-border-to-a-textblock>
* <https://stackoverflow.com/questions/49790301/how-to-detect-when-arrow-key-down-is-pressed-c-sharp-wpf>
* <https://stackoverflow.com/questions/6860282/dispatcher-begininvoke-action-with-parameters>
* …