

Hada T3: Interfaz Gráfica de Usuario

Creación de aplicaciones dotadas de GUI.

Objetivos del tema

- Conocer la arquitectura de capas **MVC**.
- Aprender a dotar de un interfaz gráfico (GUI) a nuestras aplicaciones de escritorio mediante el uso de *MVC*.
- Enlazar los conceptos de *programación dirigida por eventos*, vistos en el tema anterior, con el GUI de una aplicación de escritorio.
- Conocer la biblioteca GTK de creación de GUIs.
- Aprenderemos a crear el GUI de nuestra aplicación con la herramienta *Stetic* de MonoDevelop.

MVC I

- MVC surge junto con Smalltalk durante los años 70.
- Es aplicable al desarrollo de cualquier aplicación independientemente del lenguaje de programación elegido.
- No es necesario el uso de un lenguaje orientado a objetos para emplearlo, aunque esta metodología lo hace más sencillo.
- La idea clave de MVC consiste en dividir el código de una aplicación en capas, concretamente 3:
 1. **Modelo**
 2. **Vista**
 3. **Controlador**

MVC II

- ¿Por qué una estructura de capas?
- Cada una de estas capas puede ser sustituida en cualquier momento sin que esto afecte a las otras, p.e., podemos tener diferentes vistas para un mismo modelo, probarlas y quedarnos con la más apropiada.
- Esta división del código también garantiza mayor facilidad de portabilidad y de adaptación a los requerimientos del usuario.

MVC: **Modelo.** –Capa de la Aplicación–

- Es la representación ‘*software*’ del problema a resolver, sus datos, funciones, etc...–*personas, coches, asientos contables, carrito de la compra, elementos del carrito, etc...*–
- Proporciona los métodos necesarios para que:
 - Se puedan consultar los datos del modelo (*getters*).
 - Se puedan modificar los datos del modelo (*setters*).
- Los modelos no se comunican con las vistas, de este modo conseguimos una mayor independencia entre el código que constituye cada una de estas capas.
- Lógicamente...un modelo puede tener asociadas varias vistas (*unos mismos datos pueden ser presentados de distinta manera*).

MVC: **Vista.** –*Capa de Presentación*–

- Sirve para mostrar al usuario los ‘datos’ del modelo que le interesan en cada caso –*el nombre de una persona, el contenido del carrito, productos disponibles, etc...*–
- Una *vista* no tiene porqué ser solamente en *modo gráfico*, puede ser en *modo texto*, es una manera más de presentar la información del modelo.
- Las *vistas* se comunican con los modelos de forma bidireccional –*les solicitan información y pueden modificar la información en el modelo*–.
- En la arquitectura MVC original las vistas se pueden ‘anidar’ dando lugar a lo que se llama una vista principal –*top-view*– compuesta de subvistas. A efectos de la asignatura sólo usaremos vistas simples, no compuestas por otras.

MVC: **Controlador.** –*Capa de Interacción*–

- Contiene el código que hace de interfaz entre los dispositivos de entrada –*teclado, ratón, etc...*– y las capas de la Vista y el Modelo.
- Esta contiene el código que permite al usuario interactuar con las Vistas.
- Normalmente no tendremos que escribir el código relacionado con esta capa ya que en nuestro caso, éste, es el que proporciona la biblioteca gráfica que empleamos: Gtk+.

Biblioteca Gtk+ I

- Es la biblioteca de creación de interfaces de usuario que emplearemos: Gtk+.
- Permite crear aplicaciones de escritorio en GNU/Linux, OSX y Windows.
- También permite crear aplicaciones web de manera sencillas gracias al *backend* llamado broadway.
- Comenzó su desarrollo formando parte de la creación de la versión 1.0.0 de la aplicación Gimp.
- Se distribuye con licencia LGPL.
- Disponemos de una extensa documentación en formato electrónico que se puede consultar en línea.

Biblioteca Gtk+ II

- **Gtk+** es actualizado sistemáticamente un par de veces al año, hoy día podemos encontrarnos con las versiones **2.x.y** (en modo mantenimiento) y, la actualmente activa **3.x.y** la cual dará paso en breve a la serie **4.x.y**.
- La adaptación de **Gtk+** para **C#** se llama GtkSharp y está basada en **Gtk+ 2.x.y**.
- La documentación de **Gtk+** para **C#** la puedes encontrar en este enlace.
- Algunos ejemplos de aplicaciones hechas con **C# + Gtk#**:
 - Banshee, reproductor multimedia.
 - F-spot, gestor de álbumes de fotos.
 - El propio IDE MonoDevelop.
- Dispones de tutoriales sencillos para empezar con Gtk#. Echa un vistazo al conocido 'hola mundo' para Gtk# aquí, y aquí para aspectos más avanzados..

Biblioteca Gtk+ III

- Lo que denominamos de forma general **Gtk+** es un compendio de una serie de bibliotecas: GLib, GObject, GIO, Pango, Atk, GdkPixbuf, Gdk, Gtk.
- La estructura interna de Gtk+ es la de una jerarquía de clases formada por varios árboles (distintas raíces) con herencia simple.
- Estos árboles representan a cada una de las bibliotecas que hemos comentado antes (*glib*, *gdk*, *gtk*, etc...).

Gtk+ y programación dirigida por eventos

- El uso de Gtk+ desde C# se basa en lo que hemos visto en temas anteriores: *eventos/señales y manejadores/callbacks*.
- Los elementos de interfaz de usuario (*widgets, controles*) que proporciona Gtk+ definen una serie de eventos que pueden emitir cuando el usuario interactúa con ellos.
- Cada *evento* representa *algo* que el *widget* quiere comunicar al exterior, un ejemplo, evento Activated de la clase `Gtk.Button`:
 “Event launched when the `Gtk.Button` is activated.”
- El programador se dedica a conectar a estos eventos los métodos o funciones de su código que hacen las veces de *manejador* o *callback*.

Gtk+: Jerarquía de widgets

- En Gtk+ un *widget* normalmente solo puede contener a otro widget.
- Para crear interfaces de usuario funcionales necesitamos solventar esta limitación.
- Existe un tipo especial de widgets que son los llamados contenedores (Gtk.Container) y más concretamente, sus clases derivadas como HBox, VBox y Table.
- Visualmente no tienen ninguna apariencia, pero sí tienen como característica principal el que pueden contener más de un widget, así como gestionar el espacio que estos ocupan y decidir qué ocurre con ellos cuando cambia el tamaño de este espacio.

Gtk+: Diseñador de interfaces **Stetic**

- Viene incluido con MonoDevelop.
- Aunque no es necesario, es útil porque simplifica el trabajo de creación del interfaz de la aplicación (vista) y la conexión del mismo con el código C# (modelo) al hacer uso de clases parciales.
- En esta [guía](#) podemos ver un tutorial de cómo funciona. A grandes rasgos los pasos que lleva a cabo son:
 - a. [Crear el proyecto](#) (solución) desde el menú Archivo.
 - b. [Editamos y completamos](#) el interfaz inicial creado por MonoDevelop. Si no ves el panel de *propiedades de los widgets* debes activarlo desde: Ver ⇨ Paneles ⇨ Propiedades.
 - c. [Escribimos el código](#) del modelo que conecta con el interfaz de la aplicación.
 - d. Finalmente, compilamos y ejecutamos la aplicación.
- También es interesante que estudies este [tutorial para principiantes con Gtk#](#).