

Hada Práctica 3:

Interfaz de usuario y acceso a BB.DD. desde aplicaciones de escritorio

Departamento de Lenguajes y Sistemas Informáticos Universidad de Alicante

Objetivos de la práctica.

- Aprender a crear una solución en MonoDevelop que conste de varios proyectos.
- Aprender a crear una aplicación con interfaz gráfica de usuario haciendo uso de C# y Gtk#.
- Aprender a crear una aplicación que haga uso de un motor de BB.DD. SQL, como SQLite, desde una aplicación de escritorio.
- Continuar aprendiendo a usar git.

Interfaz de usuario y acceso a BB.DD. desde aplicaciones de escritorio

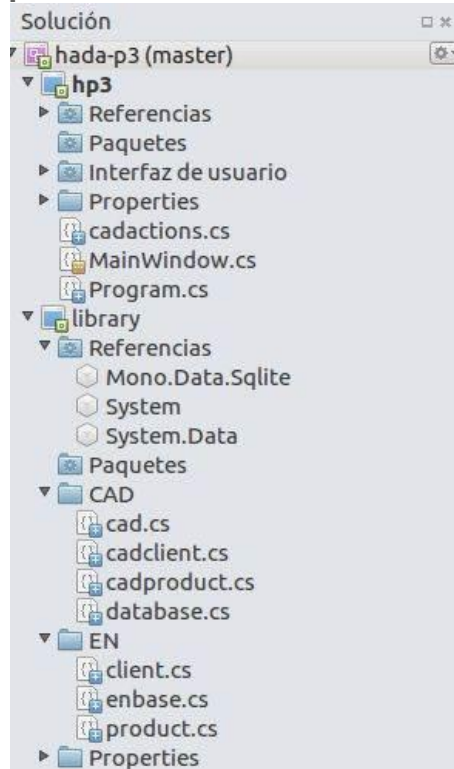
- En esta tercera práctica vamos a crear una “solución” (un contenedor de proyectos) con *MonoDevelop* para implementar una aplicación con interfaz gráfico de usuario que acceda a una BB.DD. mediante SQLite.
- Sigue los pasos indicados, **respetar el uso de mayúsculas y minúsculas** así como el **nombre de las carpetas, archivos, espacios de nombres, clases, métodos y argumentos** y el **formato de mensajes** que se te indique.
- Crea una solución de nombre **hada-p3** y un repositorio **git** en la carpeta de la solución y ve haciendo commits en él de todo lo que vayas haciendo. **Incluye un comentario con tu DNI/NIE en cada commit.**
- Al final del documento se indican las condiciones de entrega, los requisitos técnicos que debe cumplir la entrega para ser evaluada y una guía de evaluación de esta práctica.

Estructura de la “*solución*” de la práctica 3

- En esta práctica la solución (**hada-p3**) contendrá dos proyectos: **hp3** y **library**.
- El proyecto **hp3** creará el ejecutable de la aplicación con interfaz gráfico de usuario mientras que **library** contendrá el código de las entidades de negocio (*EN*) y las capas de acceso a datos (*CAD*).

Estructura de la “*solución*” de la práctica 3

- En el IDE de MonoDevelop deberás crear una estructura como ésta:



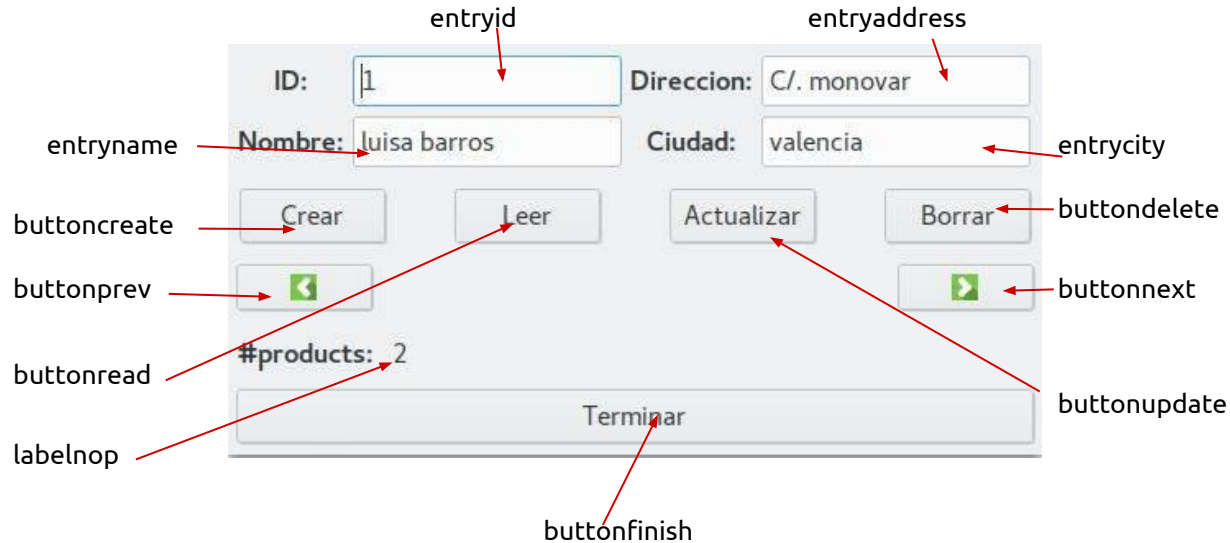
Proyecto **hp3**

- Es un proyecto de tipo **Gtk# 2.0**. Debes añadirlo a la solución.
- Su código fuente se encuentra en tres ficheros:
 - **Program.cs**: Es el programa principal que te genera, no debes modificarlo.
 - **MainWindow.cs**: Contendrá los *callbacks* asociados a los eventos que se produzcan al interactuar el usuario con el interfaz gráfico de la aplicación.
 - **cadactions.cs**: Contendrá el código para crear la BB.DD. y sus tablas, así como insertar ciertas filas en ellas.
- El código SQL para crear la BB.DD. que usaremos es el siguiente:

```
CREATE TABLE clients (id INTEGER PRIMARY KEY, name TEXT, address TEXT, city TEXT);  
CREATE TABLE products (id INTEGER PRIMARY KEY, description TEXT, prize REAL,  
                        clientid INTEGER, FOREIGN KEY (clientid) REFERENCES clients (id));
```

Proyecto **hp3**. Interfaz de usuario

- El ejecutable generado presentará una ventana como ésta al usuario:




Contenido de **hp3: MainWindow.cs I**

- Define la clase **MainWindow**, tendrá los siguientes métodos además de otros:
 - `public MainWindow(string dbfn = "")`: Constructor.
 - `dbfn`: Es el nombre del archivo de la BB.DD.
 - `public void createAndPopulateDB (string dbfn)`:
Crea la BB.DD. e inserta algunos datos en las tablas de clientes y productos. El número de clientes y el número de productos comprados por cada cliente se generarán aleatoriamente en cada ejecución.
 - `protected void createClient (object sender, EventArgs e)`:
Será llamado al pulsar el botón "**Crear**" y si `entryid` es no vacío y mayor que 0 creará un nuevo cliente en la BB.DD. con los datos del resto de campos del interfaz. Si ocurre cualquier error (excepción) informará al usuario con un mensaje: "Error creando cliente: " seguido del mensaje del sistema.


Contenido de **hp3: MainWindow.cs** II

- Tendrá los siguientes métodos públicos:
 - **protected void readClient (object sender, EventArgs e):**
Será llamado al pulsar el botón “Leer” y si **entryid** es no vacío y mayor que 0 leerá un cliente de la BB.DD. y mostrará sus datos en el resto de campos del interfaz. Si ocurre cualquier error (excepción) informará al usuario con un mensaje: “Error leyendo cliente: ” seguido del mensaje del sistema.
 - **protected void updateClient (object sender, EventArgs e):**
Será llamado al pulsar el botón “Actualizar” y si **entryid** es no vacío y mayor que 0 actualizará los datos del cliente cuyo id coincida con **entryid** en la BB.DD. con los datos del resto de campos del interfaz. Si ocurre cualquier error (excepción) informará al usuario con un mensaje: “Error actualizando cliente: ” seguido del mensaje del sistema.

Contenido de **hp3: MainWindow.cs** III

- Tendrá los siguientes métodos públicos:
 - **protected void deleteClient (object sender, EventArgs e):**
Será llamado al pulsar el botón **"Actualizar"** y si **entryid** es no vacío y mayor que 0 borrará ese cliente de la BB.DD. y mostrará "0" en **entryid** y la cadena vacía ("") en el resto de campos del interfaz. Si ocurre cualquier error (excepción) informará al usuario con un mensaje: "Error borrando cliente: " seguido del mensaje del sistema.
 - **protected void prevClient (object sender, EventArgs e):**
Será llamado al pulsar el botón  (**buttonprev**) y si **entryid** es no vacío y mayor que 1 actualizará el interfaz con los datos del cliente cuyo id coincida con **entryid-1**. Si ocurre cualquier error (excepción) informará al usuario con un mensaje: "Error leyendo cliente: " seguido del mensaje del sistema.

Contenido de **hp3: MainWindow.cs** IV

- Tendrá los siguientes métodos públicos:
 - **protected void nextClient (object sender, EventArgs e):**
Será llamado al pulsar el botón  "buttonnext" y si **entryid** es no vacío y menor que el máximo de clientes en la BB.DD. actualizará el interfaz con los datos del cliente cuyo id coincida con **entryid+1**. Si ocurre cualquier error (excepción) informará al usuario con un mensaje: "Error leyendo cliente:" seguido del mensaje del sistema.
 - **protected void updateNumberOfProducts (object sender, EventArgs e):**
Será llamado cada vez que se lea un cliente de la BB.DD. y actualizará la etiqueta **labelnop** con el número de productos comprados por ese cliente.

Contenido de **hp3: cadactions.cs**

- Declara la clase **GUI.CadActions**. El constructor por defecto de esta clase no hace nada.
- Tendrá los siguientes métodos públicos:
 - `public static void createAndPopulateDB (string dbfn):`
Este método de clase creará la BB.DD. en el fichero dbfn con las instrucciones SQL descritas anteriormente y luego insertará en ella un número aleatorio de clientes (entre 1 y 20) y para cada cliente un número aleatorio de productos (entre 1 y 5) comprados por ese cliente.

Relación entre los proyectos **hp3** y **library**

- Como te puedes imaginar el código que debe haber en los archivos del proyecto **hp3** pertenece sólo a la capa de presentación.
- Cada vez que desde el interfaz se tenga que hacer algo con `clientes` o `productos`, se deberá llamar a los métodos apropiados de la clase correspondiente de la lógica de negocios (EN).
- A su vez, cada vez que haya que hacer algo relacionado con la parte de acceso a datos habrá que llamar a los métodos apropiados de la clase correspondiente de la capa de acceso a datos (CAD).

Proyecto **library**

- Es un proyecto de tipo **Librería**. Debes añadirlo a la solución.
- Su código fuente se encuentra distribuido en carpetas:
 - **CAD**: Contiene todo el código de la capa de acceso a datos. Se distribuye en los ficheros: `cad.cs`, `cadclient.cs`, `cadproduct.cs`, `database.cs`.
Todo el código de estos archivos se encuentra bajo el *espacio de nombres* **CAD**.
 - **EN**: Contiene todo el código de la capa de la lógica de negocio. Se distribuye en los ficheros: `client.cs`, `enbase.cs`, `product.cs`.
Todo el código de estos archivos se encuentra bajo el *espacio de nombres* **EN**.

Proyecto **library**: **CAD:cad.cs**

- Define el interfaz **CAD**:

```
public interface CAD {  
    void      create (EN.ENBase en);  
    EN.ENBase read (int id);  
    void      update (EN.ENBase en);  
    void      delete (int id);  
}
```

Proyecto **library**: **CAD:database.cs** I

- Define la clase **DataBase** que tendrá, entre otros, los siguientes componentes:
 - `public enum Creation { NO, YES }`
 - `public SqlConnection connection { get { return con; } }`
 - `public DataBase (string fname, Creation create = Creation.NO)`: Crea un objeto que representa la BB.DD. SQLite que hay en el fichero `fname`. El parámetro `create` indica si se debe crear o no.
 - `~DataBase`: Cerrará la conexión con la BB.DD. si estaba abierta.
 - `private void createClientTable ()`: Creará la tabla de clientes según el esquema antes visto.
 - `private void createProductTable ()`: Creará la tabla de productos según el esquema antes visto.

Proyecto library: CAD:database.cs II

- **public void openConnection ()**: Abrirá la conexión con la BB.DD. y si no se puede capturará la excepción de tipo `SqliteException` y mostrará en el terminal la cadena "Opening connection failed.\nError: " seguida de la cadena que describe el error.
- **public void closeConnection ()**: Cerrará la conexión con la BB.DD. y si no se puede capturará la excepción de tipo `SqliteException` y mostrará en el terminal la cadena "Closing connection failed.\nError: " seguida de la cadena que describe el error.
- **public bool isOpen {...}**: Propiedad que nos dice si la conexión con la BB.DD. está abierta o no.

Proyecto **library: CAD:cadclient.cs** I

- Define la clase **CADClient**, la cual implementa el interfaz **CAD** y tendrá, entre otros, los siguientes componentes:
 - `public CADClient (string db)`: Crea la BB.DD. que se corresponde con el fichero llamado `db`.
 - `~CADClient ()`: Cierra la conexión con la BB.DD..
 - `public void create (EN.ENBase en)`: Crea un nuevo cliente en la BB.DD. con los datos del cliente representado por el parámetro `en` y si no se puede capturará la excepción de tipo `SqliteException` y mostrará en el terminal la cadena `"Client create failed.\nError: "` seguida de la cadena que describe el error.
 - `public EN.ENBase read (int id)`: Devuelve el cliente leído de la BB.DD. con `id == id` y si no se puede capturará la excepción de tipo `SqliteException` y mostrará en el terminal la cadena `"Client read failed.\nError: "` seguida de la cadena que describe el error.

Proyecto library: CAD:cadclient.cs II

- **public void update (EN.ENBase en):** Actualiza los datos de un cliente en la BB.DD. con los datos del cliente representado por el parámetro *en* y si no se puede capturará la excepción de tipo `SqliteException` y mostrará en el terminal la cadena “Client update failed.\nError: ” seguida de la cadena que describe el error.
- **public void delete (int id):** Borra el cliente de la BB.DD. con *id* == *id* y si no se puede capturará la excepción de tipo `SqliteException` y mostrará en el terminal la cadena “Client delete failed.\nError: ” seguida de la cadena que describe el error.
- **public int numberOfProducts (int id):** Devuelve el número de productos comprados por el cliente con *id* == *id* y si no se puede capturará la excepción de tipo `SqliteException` y mostrará en el terminal la cadena “Client numberOfProducts failed.\nError: ” seguida de la cadena que describe el error.

Proyecto **library**: **CAD:cadproduct.cs**

- Define la clase **CADProduct**, la cual implementa el interfaz **CAD** y tendrá, entre otros, los siguientes componentes:
 - `public CADProduct (string db)`
 - `~CADProduct ()`
 - `public void create (EN.ENBase en)`
 - `public EN.ENBase read (int id)`
 - `public void update (EN.ENBase en)`
 - `public void delete (int id)`
- Todos estos métodos hacen lo mismo y se comportan de la misma manera que los métodos correspondientes de `CADClient`, excepto que éstos trabajan con la *tabla de productos* de la BB.DD.

Proyecto **library**: **EN:enbase.cs**

- Define el interfaz **ENBase**:

```
public interface ENBase {  
    int id { get; set; }  
    string ToString ();  
}
```

Proyecto **library**: **EN:client.cs**

- Define la clase **Client**, la cual implementa el interfaz **EN** y tendrá, entre otros, los siguientes componentes:
 - `public Client (int id, string name = "", string address = "", string city = "")`
 - `public string name { get; set; }`
 - `public string address { get; set; }`
 - `public string city { get; set; }`
 - `public int id { get; set; }`
 - `public List<Product> products { get { ... } }`
 - `public void addProduct (Product p)`: Añade el producto **p** a la lista de productos comprados por este cliente.
 - `public void removeProduct (Product p)`: Elimina el producto **p** de la lista de productos comprados por este cliente.
 - `public void save (string dbname)`: Guarda este cliente en la BB.DD. almacenada en el fichero con nombre **dbname**.

Proyecto **library**: **EN:product.cs**

- Define la clase **Product**, la cual implementa el interfaz **EN** y tendrá, entre otros, los siguientes componentes:
 - `public Product (int id, int cid = 0, string description = "", double price= 0.0)`
 - `public string description { get; set; }`
 - `public double price { get; set; }`
 - `public int cid { get; set; }`
 - `public int id { get; set; }`
 - `public void save (string dbname):` Guarda este producto en la BB.DD. almacenada en el fichero con nombre `dbname`.

Entrega.

- La entrega de esta práctica consiste en el directorio de la solución hada-p3, junto con todo su contenido, comprimido en un fichero llamado hada-p3.tgz.
 - Este archivo lo puedes crear así en el terminal:
`tar cfz hada-p3.tgz hada-p3`
- **Lugar y fecha de entrega:** La entrega se realizará en <http://pracdlsi.dlsi.ua.es> en las fechas allí publicadas.
- **No se admitirá ningún otro método de entrega.**

Requisitos técnicos I.

Requisitos que tiene que cumplir este trabajo práctico para ser evaluado (si no se cumple alguno de los requisitos la calificación será **cero**):

- El archivo entregado se llama `hada-p3.tgz` (todo en minúsculas).
- Al descomprimir el archivo `hada-p3.tgz` se crea un directorio de nombre `hada-p3` (todo en minúsculas).
- Dentro del directorio `hada-p3` hay un archivo de nombre `hada-p3.sln`.
- Dentro del directorio `hada-p3` hay tres directorios: `hp3`, `library` y `.git`.
- Los directorios `hada-p3/hp3` y `hada-p3/library` contiene los archivos con el código de la práctica y se llaman como se indica en el enunciado (respetando en todo caso el uso de mayúsculas y minúsculas).

Requisitos técnicos II.

- Los nombres de espacios de nombres, clases y métodos implementados, así como sus argumentos, se llaman como se indica en el enunciado (respetando en todo caso el uso de mayúsculas y minúsculas).
- Los mensajes producidos siguen el formato especificado en el enunciado (respetando en todo caso el uso de mayúsculas y minúsculas).
- Se han realizado al menos 3 commits y en cada uno de ellos los cambios o adiciones realizados demuestran avances en el desarrollo de la práctica. Cada uno de estos commits deberá contener, al menos, tu DNI/NIE en el comentario del commit.

Guía de evaluación.

- La creación de cada una de las clases e interfaces (MainWindow, CadActions, DataBase, CadClient, CadProduct, ENBase, Client, Product) así como de todos y cada uno de sus componentes (constructores, propiedades, métodos, etc...) trabajando de forma correcta supondrá hasta el 90% de la nota.
- Los distintos commits realizados a lo largo de la creación de la práctica supondrán hasta el 10% de la nota.