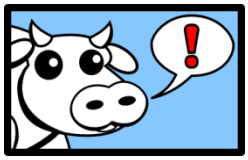
3. La Programación por Capas

¿Qué es la Programación en Capas?

Para responder esta pregunta, es necesario hacer una distinción, muy importante, entre el estilo de programación que se utilizó en cursos anteriores y el que se pretende utilizar en este curso.

En los cursos anteriores, el enfoque siempre ha sido el **comprender el proceso de programar**, las particularidades de la algoritmia y/o el diseño aplicada a **casos muy concretos**. En ese sentido, lo que hacíamos era diseñar programas **que cumplían con una determinada funcionalidad o funcionalidades**. Pero difícilmente dichos programas califiquen como ***aplicaciones***.

En este curso, el enfoque es el desarrollo de aplicaciones completas. Pero, ***¿qué es una aplicación?***



Aplicación:
Se llama "aplicación" a un programa o conjunto de programas diseñados y desarrollados específicamente para permitir a un **usuario humano final** el cumplimiento de una tarea, o tareas, **muy bien definidas**. Esto lo diferencia del **software del sistema**, por ejemplo, que en muchos casos no esta diseñado para su ejecución de forma independiente por seres humanos.

O sea, una aplicación es un conjunto de programas con un propósito muy específico: **ayudar a un ser humano en el cumplimiento de una tarea**. Y es con ese propósito que se enfoca su programación: en generar una herramienta que **permita, facilite y/o agilice el cumplimiento de dicha tarea**, mediante el uso del potencial de la computadora. Si el resultado de nuestra programación **no logra dicho objetivo**, sin dudas es un programa, pero **no es una aplicación**.

Con esto en mente, podemos explicar con mayor facilidad el concepto de la Programación en Capas, ya que tiene todo que ver con la idea de que estamos diseñando y desarrollando algo que será utilizado por un usuario.

Esto nos plantea una serie de **cuestiones organizativas** que debemos resolver a nivel del código:

- Si programamos para un usuario final, es muy posible que debamos **someter nuestro código a revisiones frecuentes** (corrección de errores, agregado de funcionalidades nuevas o mejora de las existentes, etc.), las cuales serían mucho más ágiles si el código está bien organizado. No es lo mismo buscar **una funcionalidad** entre **varios miles de líneas de código**, distribuidas en varias decenas de archivos de toda la aplicación; **que buscarla entre unos pocos archivos bien identificados**.

- Al mismo tiempo, los usuarios exigen, por motivos de comodidad y eficiencia, que nuestra aplicación **funcione de forma ágil y fluida**, con un **consumo razonable de recursos** en su dispositivo. Esto requiere de **procesos de optimización del código**, los cuales son más eficientes y eficaces si el código está bien **organizado a nivel funcional**. O sea, que las **funciones que realizan tareas similares están agrupadas en un mismo sitio**.

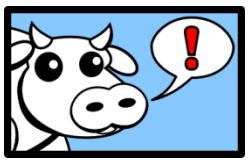
- El usuario **no necesita**, y en general, no quiere, ver **aspectos internos del proceso de nuestra aplicación**. De hecho, en muchos casos, estos aspectos **deberían estar ocultos** para el usuario por **motivos de seguridad**.

Todas estas cuestiones son las que nos llevan a desarrollar aplicaciones con el paradigma de Programación en Capas.

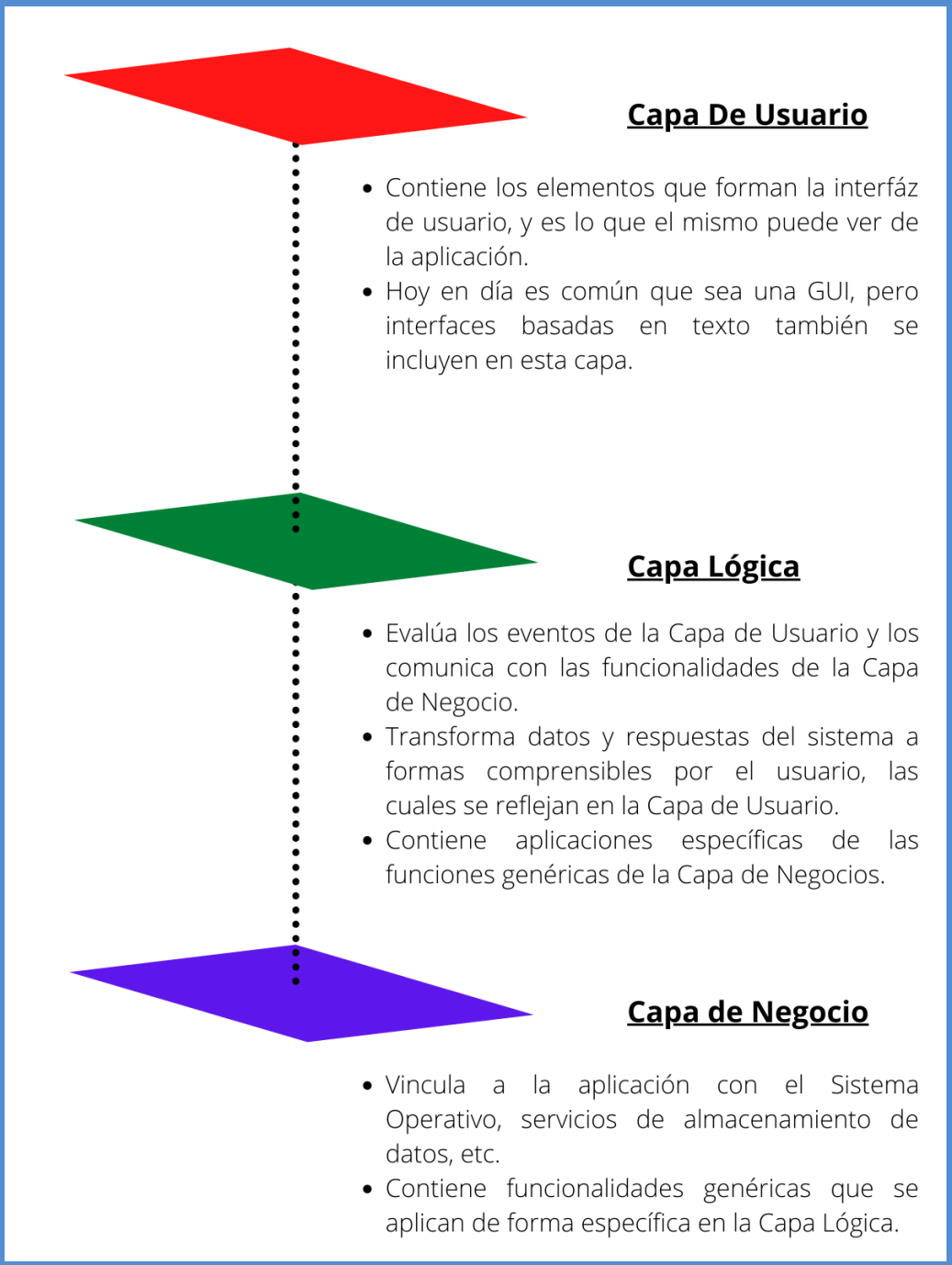
La idea básica detrás de este paradigma es que, al **agrupar la funcionalidad** de nuestra aplicación en **capas o familias de componentes funcionales**, el desarrollo de la misma es **más sencillo y escalable** (agregar una funcionalidad nueva implica la modificación de solo los componentes involucrados), obtenemos una aplicación **más fácil de mantener** (ya que todo error se encuentra localizado en componentes específicos) y el código generado resulta **más fácilmente reutilizable** en desarrollos futuros.

Este enfoque de desarrollo es propio de las arquitecturas de tipo **cliente-servidor**, ya que estas naturalmente implican el uso de múltiples niveles de manejo y procesamiento de la información.

Si bien se puede hablar de un desarrollo en ***"n capas"*** (cuando, por ejemplo, alguna de las capas es, en sí misma, desarrollada en capas), el enfoque más usual es el uso de **tres capas**:

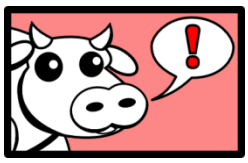


Descripción de las Capas de una Aplicación:



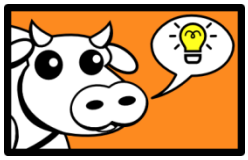
Ejemplo de una Aplicación Desarrollada en Capas: ChequeaPares

Para entender mejor **cómo funcionan** estas tres capas, **qué elementos contienen** y **cómo se relacionan** entre ellas, veamos un ejemplo con una aplicación muy sencilla que llamaremos ***"ChequeaPares"***. El propósito de esta función es muy simple: es una herramienta para comprobar si un número es par o no.



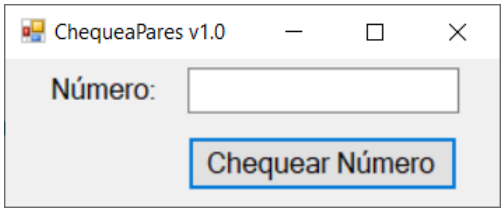
NOTA: En este momento solo nos interesa la **organización del desarrollo** de esta aplicación, por lo que **no se manejará código detallado en este ejemplo**

Empecemos por la **capa de usuario**: ¿qué es lo que un **usuario** de nuestra aplicación **vería al iniciarse la misma**? Como nuestra aplicación se ejecutaría en una PC normal (lo que en general suele llamarse una **aplicación de escritorio**), lo que el usuario vería sería una **ventana**. Y como la función expresa de nuestra aplicación es el chequear números para determinar si son pares o no, esta ventana debería contener los elementos adecuados, de forma que el usuario entienda qué clase de información se requiere para continuar. Eso implica un **campo de texto,** debidamente identificado con una **etiqueta**, donde el usuario pueda ingresar el número a chequear. Finalmente necesitamos un **botón** para que el usuario de la orden de realizar el chequeo.



NOTA: En este ejemplo **solo mostraremos la apariencia de la ventana**, pero conviene aclarar que en C# (y en cualquier otro lenguaje de programación) el **diseño de una ventana** se define en un **archivo de código**, anexo a la clase de la ventana, con extensión ***designer.cs,*** donde se declaran la ventana, los objetos de la misma y sus propiedades. Este código **se genera de forma automática** mediante un **editor visual** tanto en ***Visual Studio*** como en ***MonoDevelop***, y en general no es necesario editarlo manualmente.

El resultado sería muy similar a esto:



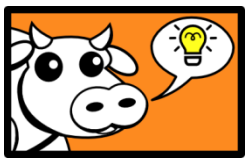
Ahora bien, la capa de usuario no se reduce a solo esta ventana (o ventanas, si nuestra aplicación requiriera más de una). También debemos incluir en esta capa **todo elemento de comunicación con el usuario**: ventanas de diálogo, menú contextual personalizado, alertas, etc.

En el caso de nuestra aplicación, debemos pensar en qué sucede al presionar el botón. Internamente se iniciaría un **proceso de chequeo** de la **información proporcionada por el usuario** en la ventana. Un análisis acerca de qué datos puede ingresar el usuario, nos deja con las siguientes posibilidades:

- El usuario ingresó un **número entero**, y este número **es par**.
- El usuario ingresó un **número entero**, y este número **no es par**.
- El usuario ingresó algo que **no es un número entero**, **no es un número** (por ejemplo, un texto) o **no ingresó nada**.

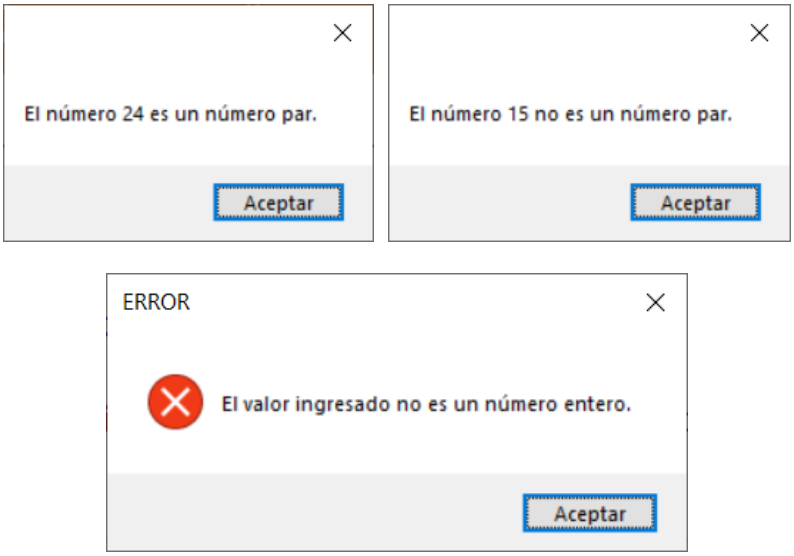
Las primeras dos posibilidades son resultados que corresponden a si el usuario ingresó la **información que el sistema espera recibir**. La última se corresponde a **un error**, ya que se trata de datos que nuestra aplicación **no puede manejar**, por lo tanto, **debe detectarlos** y **notificar al usuario** del error.

Eso nos obliga a generar varias ventanas extra para representar estas posibilidades.

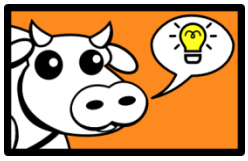


Más adelante veremos que en una aplicación es **importante comunicarse con el usuario** de forma efectiva, y que un **buen manejo de mensajes** de notificación y/o errores redunda en una **mejor experiencia del usuario** con nuestra aplicación.

Nuestras ventanas extra serían lo que conocemos como **diálogos:** ventanas que contienen un mensaje (que puede ser de confirmación o de error), y algún botón mediante los cuales el usuario puede confirmar o descartar el mensaje de la ventana.



Ahora consideremos qué contendrá la **capa lógica**. Esta capa contendrá la funcionalidad que permite que nuestra aplicación **detecte las acciones del usuario**.



Toda acción del usuario en una ventana que requiera una reacción de la aplicación debe, por Estas acciones son llamadas eventos (y las explicaremos en profundidad en el siguiente capítulo).

En C# la capa lógica abarca los **archivos de clase** tanto de las **ventanas individuales** de la aplicación, como de **cualquier otro elemento visual** que hayamos desarrollado.

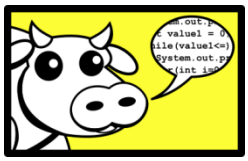
En general, en un archivo de la capa lógica debemos incluir funcionalidades que se ocupen de:

- **Eventos** que tengan lugar en las ventanas.
- **Adquisición** y **procesamiento inicial** de datos proporcionados por el usuario.
- **Inicialización** y **ejecución** de métodos contenidos en la **capa de negocios**.
- Procedimientos **internos al funcionamiento** de una ventana, pero que **no son parte de un evento particular**.

En el caso de nuestro ejemplo, la capa lógica abarcaría el archivo de clase de la ventana que diseñamos antes, y si le aplicamos el punteo que hemos planteado, debemos declarar las siguientes funcionalidades:

- El evento que se desata al **dar clic en el botón** de la ventana.
- Recopilar **el valor ingresado por el usuario** en el cuadro de texto.
- El **chequeo de ese valor** para corroborar que sea adecuado, y **mostrar el diálogo apropiado**.

El código fuente quedaría muy semejante a esto:



Ventana.cs

```
class Ventana{
      //Esta función es para chequear el valor del campo de texto
      // por si es un texto, un número no entero, o nulo.
      private bool chequearValor(TextBox campoDeTexto)
         {
            //...
         }

      //Esta función controla el evento del clic del botón
      private void boton_Click(object sender, EventArgs e)
      {
            if( chequearValor(campoDeTexto) ){
                  //...
                  //En alguna parte del código realizamos el
                  // chequeo sobre si el número es par o no, con
                  // una función de la capa de negocio.
                  if( chequearPar(valor) ){
                        MessageBox.show("Este número es par");
                  }
                  else{
                        MessageBox.show("Este número no es par");
                  }
            }
      }
}
```

En el ejemplo resaltamos una función llamada **`chequearPar()`**, que **no está declarada** en la clase de la ventana. Esto es debido a que esta función proviene de la **capa de negocios.**

La capa de negocios va a estar compuesta, al igual que la capa lógica, de clases. Estas clases se van a ocupar de todas las operaciones que impliquen un **procesamiento de datos** que sea **ajeno a la funcionalidad de las ventanas**.

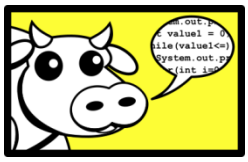
El nombre "capa de negocios" proviene, justamente, de que las funciones de esta capa, por lo general, "negocian" con **servicios provenientes del "exterior" de nuestra aplicación** (el sistema operativo, servidores de bases de datos, etc.).

Se trata, por lo tanto, de funciones que **trabajan con los datos de forma muy genérica.** Por ejemplo, si la capa lógica contiene código que realiza una **consulta específica a la base de datos**, la capa de negocios tendrá una función que **permita conectarse a la base de datos**, pero **sin especificar la consulta**. La capa de negocios tendrá funciones generales que se aplicarán de forma específica en la capa lógica.

En lo que respecta a nuestro ejemplo, la función **`chequearPar()`** es la que debe procesar el valor de un número y determinar si es par o no. Si en un futuro deseáramos ampliar la funcionalidad de nuestra aplicación, o surgiera un error, resultaría muy incómodo tener que buscar la función que realiza el chequeo específico entre todo el código de la clase de la ventana, por ejemplo. Además de que, **si tenemos todo el código** en la misma clase de la ventana, esta **va a ocupar recursos innecesarios** al ejecutarse.

En cambio, si separamos las funciones de procesamiento de datos de las que se ocupan del manejo de los elementos de la ventana, esta va a utilizar **solo los recursos que necesite, cuándo los necesite**. Esto nos fuerza a desarrollar una función **más genérica, menos dependiente de dónde debe ejecutarse**. Permitiendo, además, que sea **un código que podemos reutilizar** con mucha facilidad en otro proyecto.

Nuestra capa de negocios, entonces, sería algo así, formada por **una clase aparte** que llamaremos **`Funciones.cs`**:



Funciones.cs

```
class Funciones{
      public bool chequearPar(int numero){
            //Aqui iría el código encargado de chequear si la
            // variable 'numero' es par o no
      }
}
```

Como puede observarse, **`chequearPar()`** no hace referencia a ningún campo de texto, sino solamente a **un número entero**. Es una función **muy genérica**, ya que *"no le importa"* de dónde, ni cómo viene ese número, **eso es el trabajo de la capa lógica**, **`chequearPar()`** solamente se encarga de **procesar esa información**.

En los siguientes capítulos profundizaremos en las metodologías que usaremos para **dividir las funcionalidades entre las diferentes capas** de una aplicación, dependiendo de la naturaleza de las mismas.