UD1. CONFECCIÓN DE INTERFACES DE USUARIO

Objetivos

RA1. Generar interfaces gráficas de usuario mediante editores visuales utilizando la funcionalidade del editor y adaptando el código generado.

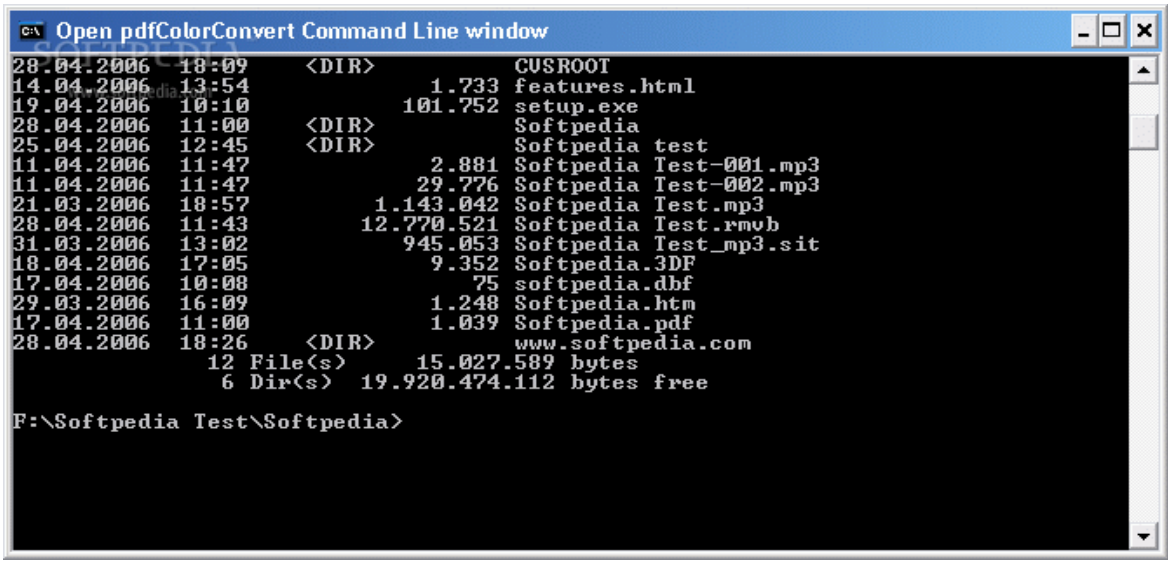
- CA1.1. Se crea una interfaz gráfica utilizando los asistentes de un editor visual.
- CA1.2. Se utilizan las funciones del editor para localizar los componentes de la interfaz.
- CA1.3. Se modifican las propiedades de los componentes para adecuar a las necesidades de la aplicación.
- CA1.4. Se analiza código generado por el editor visual.
- CA1.5. Se modifica el código generado por el editor visual.
- CA1.6. Se asocian a los eventos las acciones correspondientes.
- CA1.7. Se desarrolla una aplicación sencilla para comprobar la funcionalidad de la interfaz gráfica obtenida.

Confección de interfaces de usuario

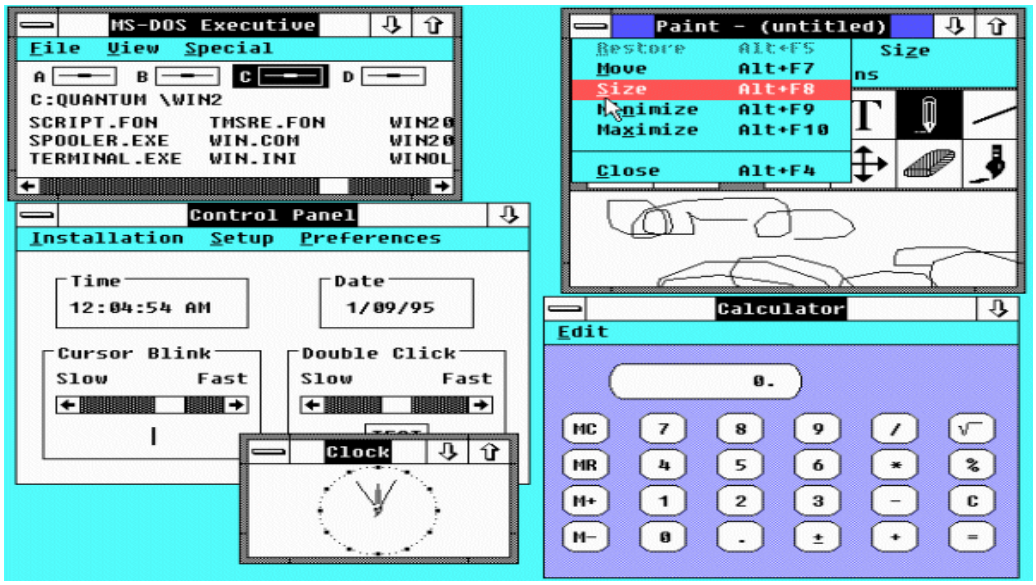
- Librerías de componentes disponibles para diferentes sistemas operativos y diversas lenguajes de programación: características.
- Herramientas propietarias y libres de edición de interfaces.
- Componentes de la interfaz visual: características y campo de aplicación. Localización e alineación.
- Unión de componentes a las orígenes de los datos.
- Asociación de acciones a eventos.
- Diálogos modales e non modales.
- Edición do código generado por la herramienta de diseño.
- Clases, propiedades e métodos.
- Eventos: escuchadores.

➔ Introducción

No fue hasta finales de la década de los sesenta cuando aparecieron las primeras interfaces gráficas de usuario tal como las entendemos hoy. Anteriormente las interfaces de usuario eran simples **CLI** (**Interfaces Command Line**) o donde se introducían las órdenes mediante comandos.

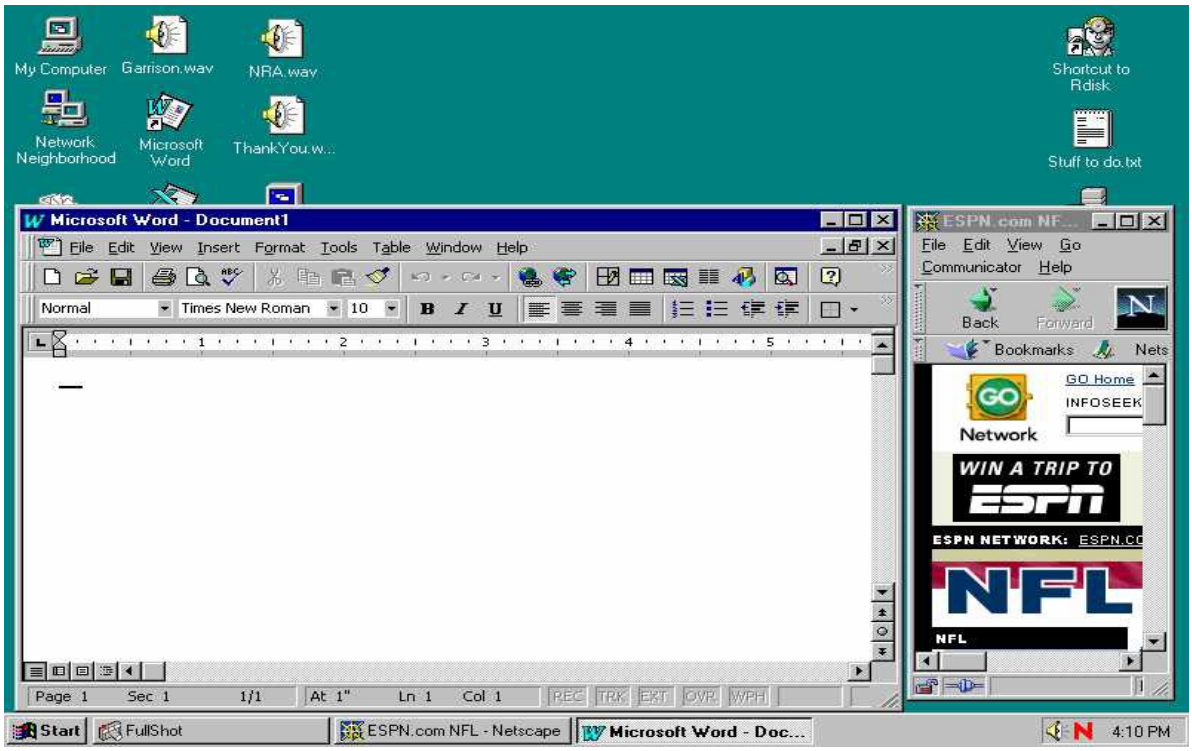


Posteriormente aparecieron **los primeros IU o Interfaces de usuario** con menús jerárquicos como por ejemplo Windows 3.1.



Más tarde en los ochenta aparecieron las **interfaces WIMP**[1] **(Windows, Icons, Menus and Pointing device**), es decir, ventanas, iconos, menus y punteros. Se mejoraron con versiones con simulación 3D y la manipulación se hace directa mediante la representación visual de los objetos y la aparición de los eventos.

1 No confundir con WYSIWYG



En el futuro se irá hacia la generación eventos mediante voz o vista e incluso el ordenador tomará sus decisiones y acciones en función de la observación del usuario.

Señalar que más del 50% del código de un programa es generado por la parte de interfaces gráficas de ahí la importancia de utilizar IDE, es decir, entornos de desarrollo que nos permite un diseño rápido de dichas interfaces.

Apunte histórico

- 1981 – Xerox Star (ventanas y 1ª aparición del ratón)
- 1984 – Apple Macintosh, X Windows (popularización del ratón)

  "There is no evidence that people want to use these things"
- 1984/7 – X Window System (X11, MIT, separado del SO y WM)
- 1985 – MS Windows (integrado en el SO e incluye WM)
- Finales de los 90 – Interfaces Web…

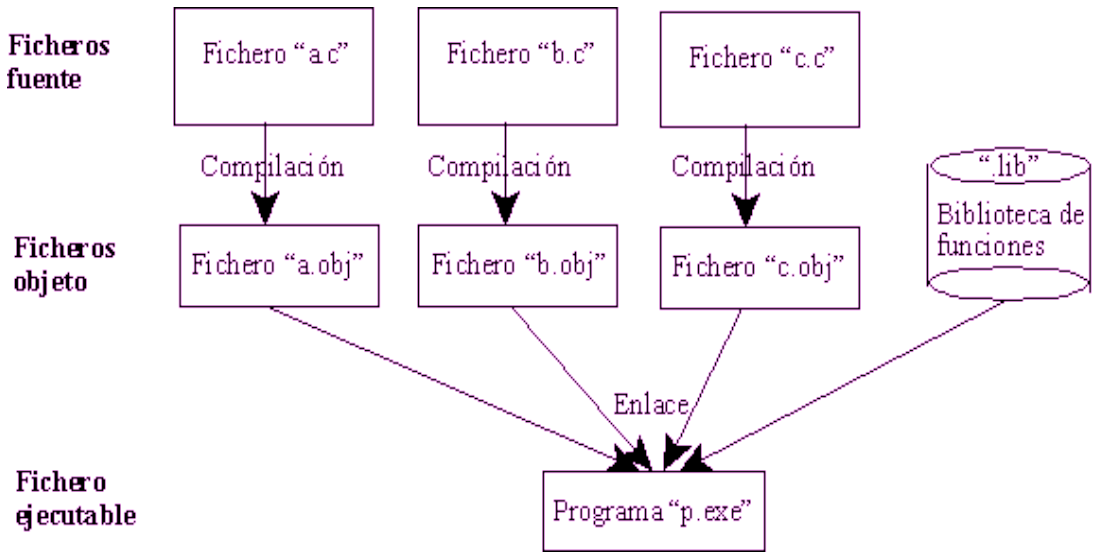
➔ Librerías de componentes disponibles para diferentes sistemas operativos y diversas lenguajes de programación: características.

Para empezar el término **librería,** en programación, es una incorrecta traducción del término inglés *library* y que en realidad hace referencia al término en castellano de *biblioteca.*

Una **biblioteca** es un conjunto de implementaciones o **subprogramas** con una interfaz bien definida para ser invocados y que son utilizados para desarrollar software. Las bibliotecas contienen **código y datos,** que **proporcionan servicios a programas independientes**, es decir, pasan a formar parte de estos. Esto permite que el código y los datos **se compartan y puedan modificarse de forma modular.** Ejecutables y bibliotecas hacen referencias (llamadas **enlaces**) entre sí a través de un proceso conocido como **enlace**, que por lo general es realizado por un software denominado **enlazador.**

A diferencia de un programa ejecutable, el comportamiento que implementa una biblioteca **no espera ser utilizada de forma autónoma** (un programa sí: tiene un punto de entrada principal), sino que su fin es **ser utilizada por otros programas, independientes y de forma simultánea**. Por otra parte, el comportamiento de una biblioteca no tiene porqué diferenciarse en demasía del que pudiera especificarse en un programa. Es más, unas bibliotecas pueden requerir de otras para funcionar, pues el comportamiento que definen refina, o altera, el comportamiento de la biblioteca original; o bien la hace disponible para otra tecnología o lenguaje de programación.

La mayoría de los sistemas operativos modernos **proporcionan bibliotecas que implementan los servicios del sistema**. De esta manera, estos servicios se han convertido en una "*materia prima*" que cualquier aplicación moderna espera que el sistema operativo ofrezca (por ejemplo el módulo de impresión). Como tal, la mayor parte del código utilizado por las aplicaciones modernas se ofrece en estas bibliotecas.



Podemos dividir las librerías en dos tipos:

- **Librerías estáticas:** es aquella que se enlaza en **tiempo de compilación**. La ventaja de este tipo de enlace es que hace que un programa no dependa de ninguna biblioteca (puesto que las enlazó al compilar), haciendo más fácil su distribución. Su inconveniente es que los **programas son más pesados y menos flexibles** a la hora de modifica su código.

  El enlazado permite al programador y al propio sistema operativo dividir un programa en varios archivos llamados **módulos**, que pueden ensamblarse por separado y enlazarse en una ocasión posterior, el enlace puede ser de naturaleza estática o dinámica. El enlace estático da como resultado, un archivo ejecutable con todos los símbolos y módulos respectivos incluidos en dicho archivo.
- **Librería dinámicas:** es aquella enlazada cuando un determinado programa se ejecuta. La ventaja de este tipo de enlace es que el programa es más liviano, y que evita la duplicación de código (por ejemplo, cuando dos programas requieren usar la misma biblioteca, se necesita sólo una copia de ésta).

  Las bibliotecas de enlace dinámico, o bibliotecas compartidas, suelen encontrarse en **directorios específicos del sistema operativo**, de forma que, cada vez que un programa necesite usar alguna, el sistema operativo conozca el lugar en el que se encuentra, para así poder enlazarla. Esto ocasiona algunos problemas de **dependencias**, principalmente entre diferentes versiones de una misma biblioteca. En Windows las librerías dinámicas se denominan **DLL (*Dinamic-Link Library)***

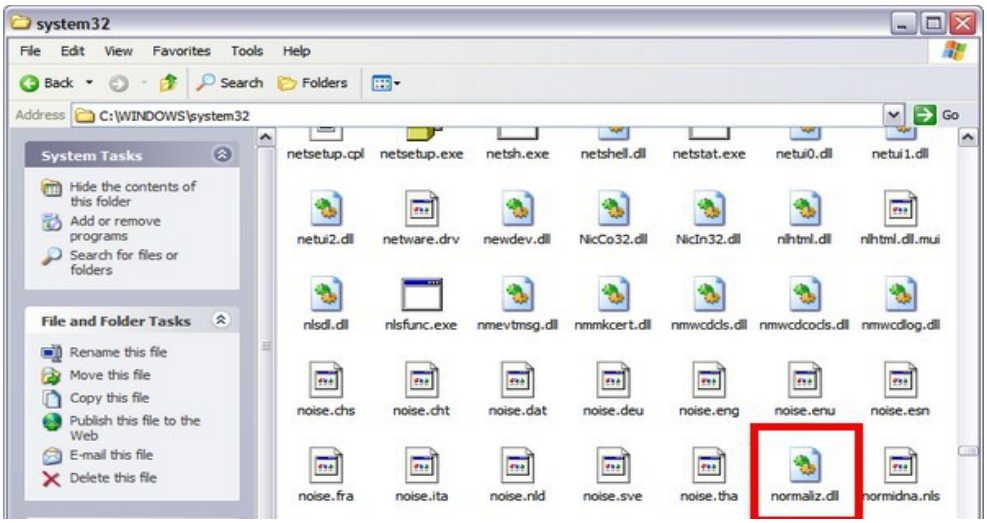
  Muchos programas tienen procedimientos a los que no llaman, salvo en circunstancias excepcionales. Haciendo uso de bibliotecas de enlace dinámico, después del ensamblaje, podemos enlazar cada procedimiento en el momento en que es llamado y cuando es necesario.

  Una de las mayores desventajas del enlace dinámico es que el funcionamiento correcto de los ejecutables **depende de una serie de bibliotecas almacenadas de forma aislada.** Si la biblioteca es borrada, movida o renombrada, o si una versión incompatible de DLL es copiada en una ubicación que aparece antes en la ruta de búsqueda, el ejecutable no se podrá cargar.

Los tipos de librerías se pueden clasificar según varios criterios, por ejemplo según el lenguaje de programación. Como será el tipo de criterio que utilizaremos solo haremos referencia a los lenguajes de uso más habitual y a las librerías relacionadas con el contenido del módulo ya que en algún caso el número es muy elevado:

- **C/C++:** podemos dividirlas en:

- **Estáticas**: Denominadas también librerías-objeto, son colecciones de ficheros objeto (compilados) agrupados en un solo fichero de extensión **.lib, .a,** etc. junto con uno o varios ficheros de cabecera (generalmente **.h).** Estas últimas contienen los ficheros estándar *stdio.h, math.h, string.h... y la graphics.h* que aún no siendo considerada como estándar está incluida en los compiladores Borland C++ para el desarrollo de aplicaciones.
- **Dinámicas:** conocidas como **DLL's,** acrónimo de su nombre en inglés ("Dynamic Linked Library"). Estas librerías se utilizan mucho en la programación para el SO Windows. Este Sistema contiene un gran número de tales librerías de terminación .DLL, aunque en realidad pueden tener cualquier otra terminación .EXE, .FON, .BPI, .DRV etc. Cualquiera que sea su terminación, de forma genérica nos referiremos a ellas como DLL's, nombre por el que son más conocidas. Suelen encontrarse en el carpeta **system32.** En Linux en el directorio **/lib.** La falta o fallo de alguna de ellas necesaria impide la ejecución de uno o más programas.



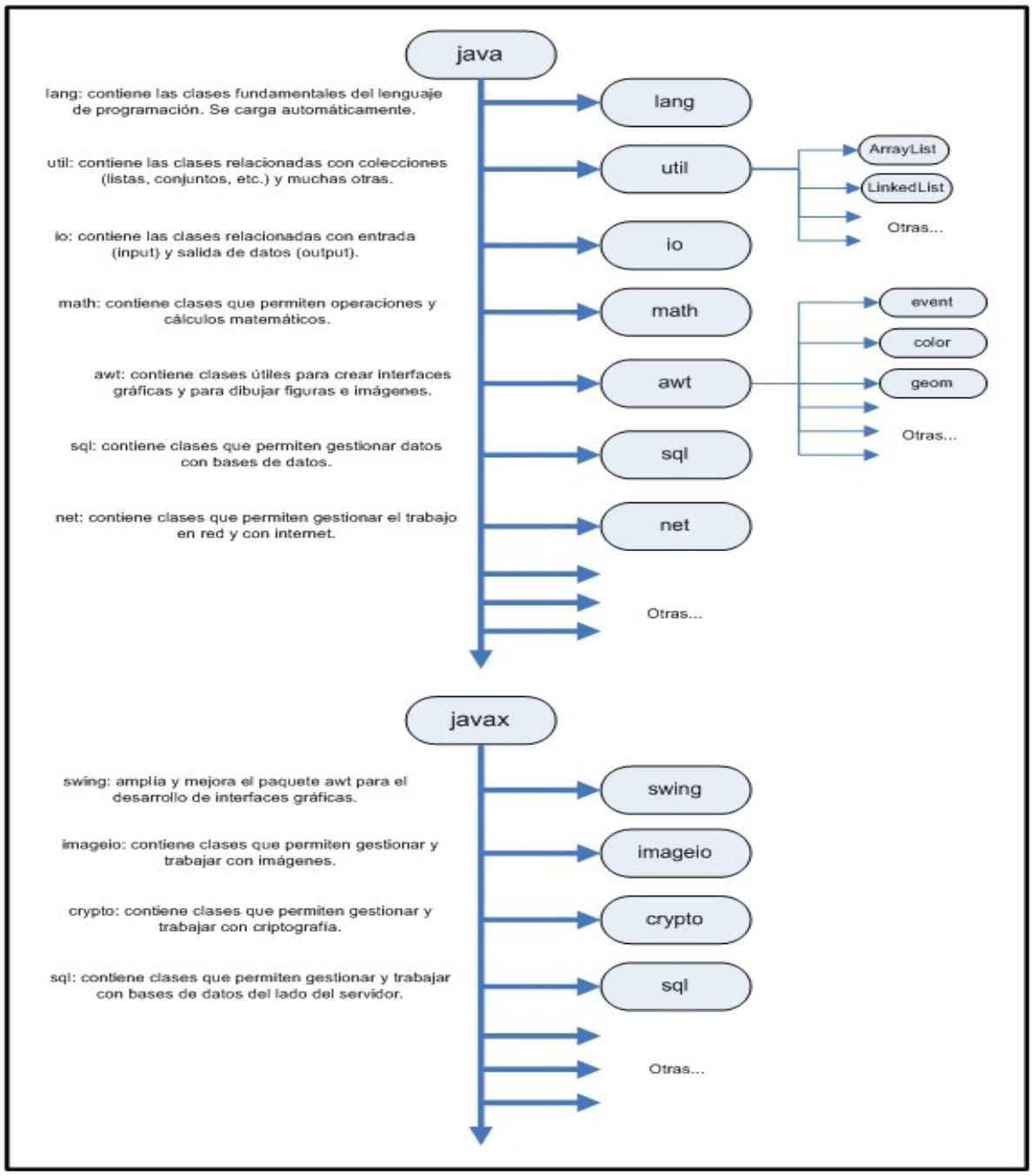
- **Java:** La biblioteca estándar de Java está compuesta por cientos de clases como System, String, Scanner, ArrayList, HashMap, etc. que nos permiten hacer casi cualquier cosa. nNnos centraremos en las más interesantes para nosotros, es decir, aquellas relacionadas directamente con la creación de entorno gráficos de usuario son:
  - **AWT (Abstract Window Toolkit):** esta contenida en la rama **java de la API del lenguaje** (ver figura siguiente) permite desarrollar interfaces de usuario gráficas. Es la librería básica y aunque ahora se usa más la Swing.

Esta compuesta por:

- Los Componentes (java.awt.Component), como los Buttons, Labels,..

- Los Contenedores (java,awt.containers), contienen componentes.
- Los gestores de posición (java.awt.LayoutManager), que posiciona loscomponentes dentro de los contenedores.
- Los eventos (java.awt.AWTEvent), que nos indican las acciones del usuario.

- **Javax.swing:** proporciona una serie de clases e interfaces que amplían la funcionalidad del anterior. Es la versión más moderna del API de Java. Están escritos en Java y son independientes de la plataforma.



- **XML:** es un lenguaje de marcas desarrollado por el World Wide Web (W3C) utilizado para almacenar datos en forma legible. Cualquier procesador de texto, que sea capaz de producir archivos .txt es capaz de generar documentos en XML, aunque en los entornos de desarrollo, como Eclipse, Visual Studio o Netbeans, reconocen los formatos y ayuda a generar un XML bien formado gracias a las librerías que lo implementan.
  De hecho XML no ha nacido sólo para su aplicación en Internet, sino que se propone como un estándar para el **intercambio de información estructurada entre diferentes plataformas.** Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.
  Entre las librerías y herramientas más importantes relacionadas con XML tenemos:
  - **Estándares de parseo (analizadores sintácticos de documentos XML)**
    - Se recomienda utilizar el estándar **DOM** con documentos XML pequeños, cuando queramos analizar el documento múltiples veces o editarlo, ya que se encuentra cargado en memoria, o queramos generar un documento XML desde cero.
    - Se recomienda utlizar el estándar **SAX** con documentos XML grandes, cuando queramos analizar el documento una sola vez o por partes (capturando los elementos importantes). También se usa cuando no se requiere una modificación estructural.
  - **Análisis del XML**
    - **Xerces2** permite el procesamiento de documentos XML tanto con el estándar DOM o con el estándar SAX, y errores. Además, integra a otras librerías independientes de parseado.
    - **JDOM** permite leer, escribir, crear y manipular ficheros XML de forma sencilla e intuitiva. Está totalmente programada en Java, lo que le permite utilizar las capacidades particulares del lenguaje, simplificando significativamente su manejo para programadores expertos en Java. Se basa en el procesamiento de un documento XML y la construcción de un árbol. Una vez construido el árbol se puede acceder directamente a cualquiera de sus componentes.
    - **JAXP (Java API for XML Processing)** permite procesar tanto el estándar DOM como el estándar SAX. Este API está diseñado para ser flexible y uniformar el desarrollo de aplicaciones Java con Xml. Además, proporciona una capa intermedia

que nos permite usar cualquier analizador XML compatible dentro de nuestra aplicación, reduciendo el acoplamiento de los coponentes de la aplicación con la implementación del analizador.

- **Serialización del XML**
  - Se recomienda utilizar las librerías **Xerces2, Xstream o JAXB** para serializar objetos Java en un medio de almacenamiento, como puede ser un archivo o un buffer de memoria, con el fin de transmitirlo a través de una conexión en red, ya sea como una serie de bytes o usando un formato humanamente más legible, como XML o JSON, entre otros. La serie de bytes o el formato empleado para la transmisión pueden ser usados para crear un nuevo objeto que es idéntico en todo al original, incluido su estado interno. Por tanto, el nuevo objeto es un CLON del original.
  - La **serialización** es un mecanismo ampliamente usado para transportar objetos a través de una red, para **hacer persistente un objeto en un archivo o base de datos,** o para distribuir objetos idénticos a varias aplicaciones o localizaciones. EL XML es un formato estándar de documentos basados en texto para almacenar datos legibles por aplicaciones que proporciona un fácil procesamiento. Para llevar a cabo este proceso se han de definir mediante reglas tanto el proceso de transformación de objetos Java a XML como el proceso de transformación inversa.
- **Transformaciones a XML**
  - En ocasiones es necesario permitir realizar transformaciones de documentos XML con lenguajes basados en XSL (XSLT, XPath). Un documento XML puede ser transformado en distintos formatos, como HTML, o en otro documento XML. Para realizar estas transformaciones basadas en XSLT, se recomienda el empleo de la librería **Xalan**.
- **Vinculación con objetos Java**
  - Por la influencia del lenguaje Java en la programación actual usando **JiBX, JAXB o XMLBeans** podemos vincular datos en XML con objetos Java. Con cualquiera de las tres librerías podemos, o partir de un esquema XML generar código Java. Se recomienda utilizar una de estas tres librerías para la vinculación de los documentos XML con los objetos Java.

* Herramientas propietarias y libres de edición de interfaces:

Software libre, con acceso a su código, generalmente gratuito:

- **Ventajas**
  - Existen aplicaciones para todas las plataformas (Linux, Windows, Mac Os).
  - El precio de las aplicaciones es mucho menor, la mayoria de las veces son gratuitas.
  - Libertad de copia.
  - Libertad de modificación y mejora.
  - Libertad de uso con cualquier fin.
  - Libertad de redistribución.
  - Facilidad a la hora de traducir una aplicacion en varios idiomas
  - Mayor seguridad y fiabilidad.
  - El usuario no depende del autor del software
- **Inconvenientes:**
  - Algunas aplicaciones (bajo Linux) pueden llegar a ser algo complicadas de instalar.
  - Inexistencia de garantía por parte del autor.
  - Interfaces gráficas menos amigables.
  - Menor compatibilidad con el hardware.

Sofware privativo (no propietario mala traducción) no permite acceso al código, aunque puede ser también gratuito.

- **Ventajas**
  - Facilidad de adquisición (puede venir preinstalado con la compra del pc, o encontrarlo fácilmente en las tiendas).
  - Existencia de programas diseñados específicamente para desarrollar una tarea.
  - Las empresas que desarrollan este tipo de software son por lo general grandes y pueden dedicar muchos recursos, sobretodo económicos, en el desarrollo e investigación.
  - Interfaces gráficas mejor diseñadas según la apreciación, subjetiva, de algunos usuarios.
  - Mayor compatibilidad con el hardware.

- **Inconvenientes**
  - No existen aplicaciones para todas las plataformas (Windows y Mac OS).
  - Imposibilidad de copia.
  - Imposibilidad de modificación.
  - Restricciones en el uso (marcadas por la licencia).
  - Imposibilidad de redistribución.
  - Por lo general suelen ser menos seguras.
  - El coste de las aplicaciones es mayor.
  - El soporte de la aplicación es exclusivo del propietario.
  - El usuario que adquiere software propietario depende al 100% de la empresa propietaria.

Para desarrollar software por su alta productividad se utilizan entornos de desarrollo o IDE. Un **entorno de desarrollo (IDE)** suele tener los siguientes componentes aunque no necesariamente todos:

- Un editor de texto
- Un compilador
- Un intérprete
- Un depurador
- Un cliente
- Posibilidad de ofrecer un sistema de control de versiones.
- Facilidad para ayuda en la construcción de interfaces gráficas de usuario. Esto últimos es lo que más nos atañe.

No voy a proponer un extenso listado de entornos de desarrollo sino los más conocidos por su frecuencia de uso. Muchos usuario opinan que el uso de estos entornos “ensucian” con líneas de código redundantes la implementación de la aplicación.

Algunos de los entornos de desarrollo de interfaces gráficas

- **Microsoft Visual Studio** es un entorno de desarrollo integrado para sistemas operativos Windows. Soporta varios lenguajes de programación tales como Visual C++, Visual C#, Visual J#, y Visual Basic .NET, al igual que **entornos de desarrollo web** como ASP.NET. aunque actualmente se han desarrollado las extensiones necesarias para muchos otros. Es un producto comercial aunque Microsft tiene versiones Express Edition gratuitas pero

no libres con ciertas limitaciones en la explotación de las aplicaciones a desarrollar. Adicionalmente, Microsoft ha puesto gratuitamente a disposición de todo el mundo una versión reducida de SQL Server o Express Edition cuyas principales limitaciones son que no soporta bases de datos superiores a 4 GB de tamaño, únicamente se ejecuta en un procesador y emplea 1 GB de RAM como máximo, y no cuenta con el Agente de SQL Server.

Actualmente está en su versión 2013.

- **Gambas** es un lenguaje de programación libre **derivado de BASIC** (de ahí que Gambas quiere decir Gambas Almost Means Basic). Es distribuido con licencia GNU-GPL. Cabe destacar que presenta ciertas similitudes con Java ya que en la ejecución de cualquier aplicación, se requiere un conjunto de librerías interprete previamente instaladas **(Gambas Runtime)** que entiendan el bytecode de las aplicaciones desarrolladas y lo conviertan en código ejecutable por el computador. Por otro lado, es posible desarrollar grandes aplicaciones en poco tiempo.
  Permite crear formularios con botones de comandos, cuadros de texto y muchos otros controles y enlazarlos a bases de datos como MySQL, Postgree o SQLite además de facilitar la creación de aplicaciones muy diversas como videojuegos (utilizando OpenGL), aplicaciones para dispositivos móviles (en desarrollo pero muy avanzado), aplicaciones de red (con manejo avanzado de protocolos HTTP, FTP, SMTP, DNS), entre otras .
- **Glade**: es una herramienta de desarrollo visual de interfaces gráficas mediante **GTK/GNOME.** Es independiente del lenguaje de programación y no generando código fuente sino **un archivo XML. GtkBuilder** es un formato XML que Glade usa para almacenar los elementos de las interfaces diseñadas. Estos archivos pueden emplearse para construirla en tiempo de ejecución mediante el objeto GtkBuilder de GTK+. GladeXML era el formato que se usaba en conjunto con la biblioteca libglade. Su conexión con lenguajes como Phyton o el IDE Anjuta (compilador C/C++) permite el desarrollo de interfaces en el mundo GNOME. Su sinónimo para KDE es **QTCreator**
- **Embarcadero Delphi**, antes conocido como **CodeGear Delphi**, **Inprise Delphi** y **Borland Delphi**, es un entorno de desarrollo de software diseñado para la programación de propósito general con énfasis en la programación visual. En Delphi se utiliza comolenguaje de programación una versión moderna de Pascal llamada Object Pascal.
- **NetBeans y Eclipse:** ambos son IDES para JAVA. El primero es de código abierto y admite más de un lenguaje de programación como PHP y Phyton. El segundo aunque gratuito no soporta licencia GNU-GPL
- **Oracle Database:** un entorno que permite la creación de bases de datos y con una

herramienta **OracleDesigner** que permite crear las interfaces para acceder a ellas. Tiene una versión gratuita **Lite.**

- **Anjuta:** de gran influencia en el mundo Linux. Es un (IDE) para programas en los lenguajes C, C++, Java, **Python** y Vala, en sistemas GNU/Linux y BSD. Su principal objetivo es trabajar con **GTK** y en el **Gnome**, además ofrece un gran número de características avanzadas de programación. Es software libre y de código abierto, disponible bajo la Licencia Pública General de GNU.
- **Diseñadores web:** Por el lado comercial está **Dreamweaver** que permite no solo el diseño de interfaces web en HTML, XML... sino el desarrollo de aplicaciones basadas por ejemplo en PHP. Soporta ASP.NET, JavaScript, CSS, ColdFusion.... No comercial también tenermos **Microsoft FrontPage.**

  Más austeros pero de igual capacidad y libres tenemos **Amaya, Kompozer, Aptana y NVU** entre otros.

Hay bastantes más herramientas de desarrollo que pemite la creación de ventanas y entornos gráficos. Pero estás, en el mercado actual, son las de mayor implantación y/o interés.

* Componentes de interfaz visual:Caracteristicas y campo de aplicación:

Las **interfaces visuales** de un programa es un conjunto de elementos hardware y software de una computadora que **presentan información al usuario** y le permiten interactuar con dicha información y con la computadora.

Existen una serie principios generales que deben acompañar al diseño e implementación de **Interfaces de Usuario**(IU), ya sea para las IU gráficas, como para la Web, que están basadas en que sean:

- **Sencilla**
- **Intuitiva**
- **Coherente**
- **Clara**
- **Predecible**
- **Flexible**
- **Consistente**

Dentro del diseño de la interacción usuario-computador se tienen en cuenta una seria de disciplinas como la psicología, filosofía, ciencia cognitiva, ergonomía, ingeniería, sociología,

antropología, lingüística y documentación entre otras.

Los elementos básicos de una interfaz gráfica son:

- **Componentes GUI (widgets)**
  - Objetos visuales del interfaz
  - Un programa gráfico es un conjunto de componentes anidados: ventanas, contenedores, menús, barras, botones, campos de texto, etc.
- **Disposición (layout): cómo se colocan los componentes para lograr un GUI cómodo de utilizar**
  - Layout managers: Gestionan la organización de los componentes gráficos de la interfaz
- **Eventos: interactividad, respuesta a la entrada del usuario :**
  - Desplazamiento del ratón, selección en un menú, botón pulsado, etc.
- **Creación de gráficos y texto - Bibliotecas Graphics**
  - Define fuentes, pinta textos,
  - Para dibujo de líneas, figuras, coloreado,...

El **escritorio es el contexto más global** dentro de la interfaz gráfica de usuario ya que representa el espacio donde se mueve y administra la información. En base a este concepto se agrupan los demás, como las carpeta, documentos y herramientas en general.

La **metáfora del escritorio** es un excelente recurso en que el usuario puede, de forma intuitiva, relacionar a través de signos o más bien **representaciones simbólicas**, qué tipo de elemento es y cuál es la acción que puede realizar. Esta metáfora en ampliamente utilizada por la mayoría de los sistemas operativos modernos que trabajan con interfaces gráficas; como Windows, Mac OS S, Linux y similares a Unix. Sus elementos están en constante evolución, acondicionamiento y acoplamiento a la semántica humana.

La metáfora de escritorio trata al monitor **como si fuera el escritorio físico del usuario**, sobre el cual pueden ser colocados los objetos tales como documentos y carpetas de documentos. Un documento puede ser abierto en una ventana, que representa **una copia de papel del documento** colocada en el escritorio. También están disponibles pequeñas aplicaciones llamadas **accesorios de escritorio**, como por ejemplo una calculadora o una libreta de notas, etc

Es de suma importancia señalar que algunos sistemas actuales, específicamente **Windows 8 y su interfaz metro**, buscan romper con este paradigma, es decir, el escritorio aun existe dentro del

sistema operativo pero funciona como una aplicación más, dejando de ser el protagonista al momento de definir nuestro espacio de trabajo como lo había sido hasta no mucho tiempo.

* Enlace de componentes a origenes de datos:

Los datos son el corazón de todas las interfaces de usuario. Desde las reservas de hotel hasta la consulta del mercado de valores, las interfaces de usuario proporcionan una forma de visualizar e interactuar con alguna forma de datos. La elección de los componentes de la interfaz de usuario que van a estar visibles y la forma de disponerlos para que proporcionen un flujo de trabajo útil **dependen principalmente de la naturaleza de los datos con los que se va a trabajar.**

Puede que la aplicación trabaje con:

- ***orígenes de datos internos,*** quizá realizando cálculos con números que un usuario ha escrito en un formulario por ejemplo una calculadora.
- ***orígenes de datos externos*****,** como bases de datos, fuentes o servicios Web, o archivos locales que contengan información.

Por otro lado, las aplicaciones podrían necesitar obtener acceso a orígenes de datos tanto internos como externos, por ejemplo, admitir dos tipos de orígenes de datos externos: XML y objetos CLR. En este caso un archivo XML local o remoto que puede suministrar datos en formato XML a la aplicación, puede usar un archivo XML que haya agregado al proyecto o puede establecer el origen de datos en la dirección URL de un archivo XML de un sitio Web.

El **enlace de datos** es el **proceso de conectar los elementos de un origen de datos a los componentes de la interfaz de usuario (controles).** Esto significa que **cada vez que cambien los datos, los componentes de la interfaz reflejarán de manera opcional dichos cambios, y viceversa.** El ejemplo más sencillo de enlace de datos sería un control de barra de desplazamiento enlazado internamente al ancho de un rectángulo. Al mover la barra de desplazamiento, el tamaño del rectángulo se agranda o se reduce a escala.

Un enlace se construye, básicamente, entre un ***origen*** **y un** ***destino***. El origen suele ser un origen de datos u otro control, y el destino siempre es un control. En el ejemplo de la barra de desplazamiento, el origen es la propiedad *Value* del control de barra de desplazamiento y el destino es la propiedad *Width* del rectángulo.

El **flujo de datos** se define como la **dirección en la que fluyen los datos entre el origen y el destino**. En el caso de la barra de desplazamiento que escala un rectángulo, sólo se necesita un enlace en una dirección: desde la barra de desplazamiento (origen) al rectángulo (destino). La mayoría de los IDES ofrecen las siguientes configuraciones de enlace para el flujo de datos:

- **OneWay** Los cambios que se realizan en el origen actualizan automáticamente el destino, pero los cambios en el destino no actualizan el origen. Por ejemplo, cuando elegimos España automáticamente se vuelquen las provincias en un *comboBox*
- **TwoWay:** Los cambios que se realizan en el origen actualizan automáticamente el destino, y viceversa. Por ejemplo, cuando cambiamos el *IVA* en un textBox para modificar lo en la base de datos automáticamente se cambie el valor del subtotal de todas las facturas que se visulicen.
- **OneWayToSource:** Ésta es la opción opuesta a OneWay y con ella se actualiza automáticamente el origen tras modificar el destino. Esta opción es útil en casos especiales en los que la propiedad de destino no está visible en el panel Propiedades, lo que puede ocurrir si no se trata de una propiedad de dependencia. El enlace OneWayToSource: permite configurar el enlace de datos en el destino.
- **OneTime:** Provoca una única inicialización del origen al destino, pero los cambios posteriores en el origen no actualizan el destino.

Para finalizar comentar, para enlaces externos a datos, los dos estándares más utilizados por los desarrolladores de aplicaciones: el **ODBC** creado por Microsoft y **JDBC** para aplicaciones den JAVA. Ambas son estándares que permite, independientemente de la estructura, crear una interfaz homogénea para el acceso a la base de datos. Permiten acceder a cualquier dato desde cualquier aplicación, sin importar qué sistema de gestión de bases de datos (DBMS) almacene los datos sin que el desarrollador tenga que preocuparse de como lo hace simplemente realiza la conexión.
Hablaremos de ellos en unidades posteriores.

* Asociación de acciones a eventos:

La programación de aplicaciones para un GUI implicaron un cambio radical de filosofía y estructura a la hora de codificar los programas.
Un **programa tradicional** tiene una estructura “lineal”, con el código repartido en una serie de funciones u operaciones. Existe **una función u operación principal o *main*** donde comienza la ejecución y a partir de ahí se encadenan las llamadas de unas funciones a otras hasta que en un punto determinado acaba la ejecución
En cambio la programación en GUIs (Interfaz Gráfica de Usuario) esta orientada a **eventos**. La mayoría de los eventos son sucesos **asíncronos** producidos por la interacción del usuario con la aplicación, y están ligados a algún elemento de la interfaz. Algunos ejemplos son:

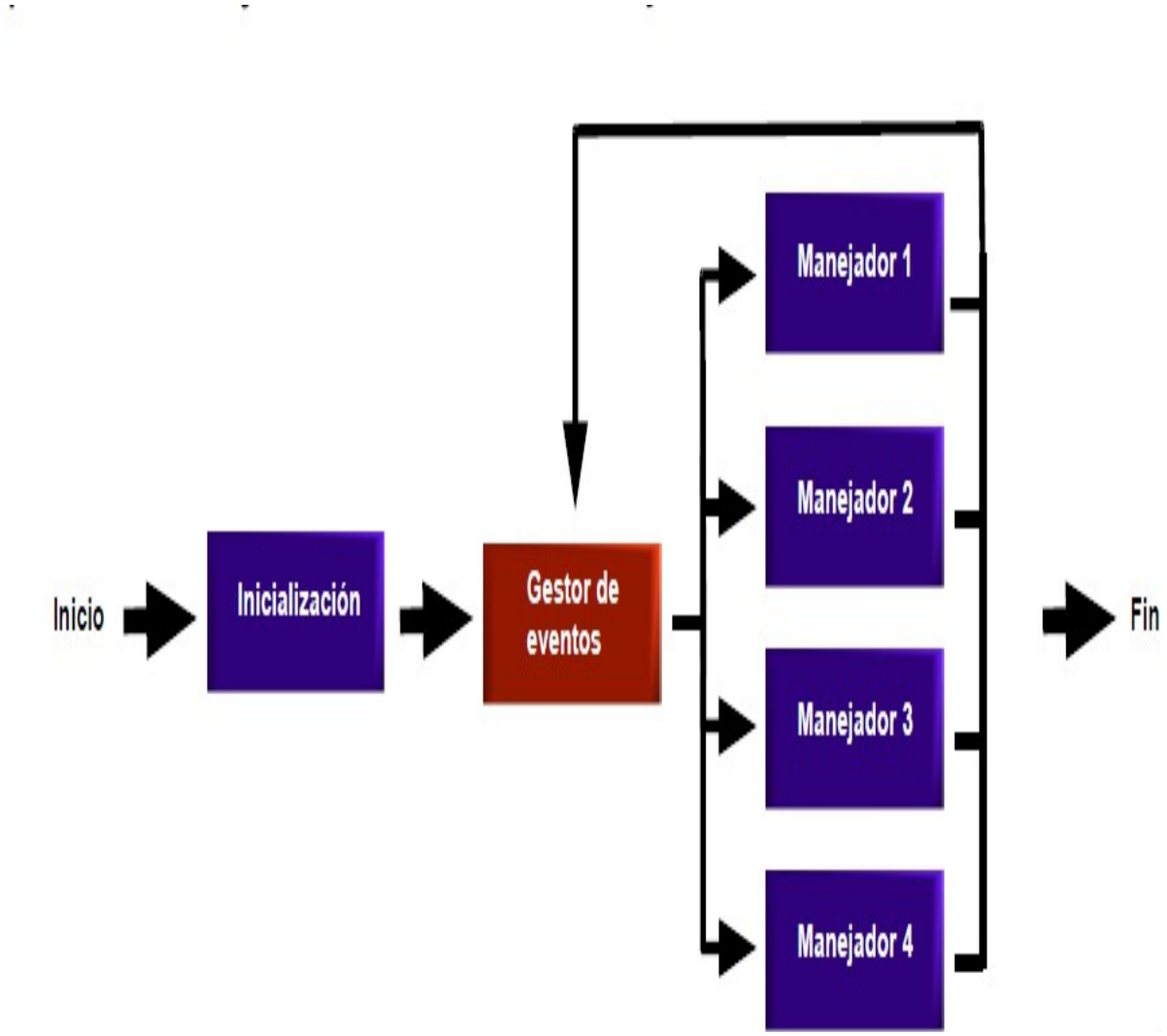
- Pulsar un botón
- Cambiar el tamaño de una ventana

- Mover una barra de desplazamiento
- Pulsar una tecla
- Tocar alguno de los botones minimizar/maximizar/cerrar de la ventana
- Hacer un click de ratón sobre un elemento determinado

Algunos eventos no relacionados directamente con el usuario y **generalmente síncronos** son:

- Aparición de una ventana
- “Tick” de un reloj programado con antelación

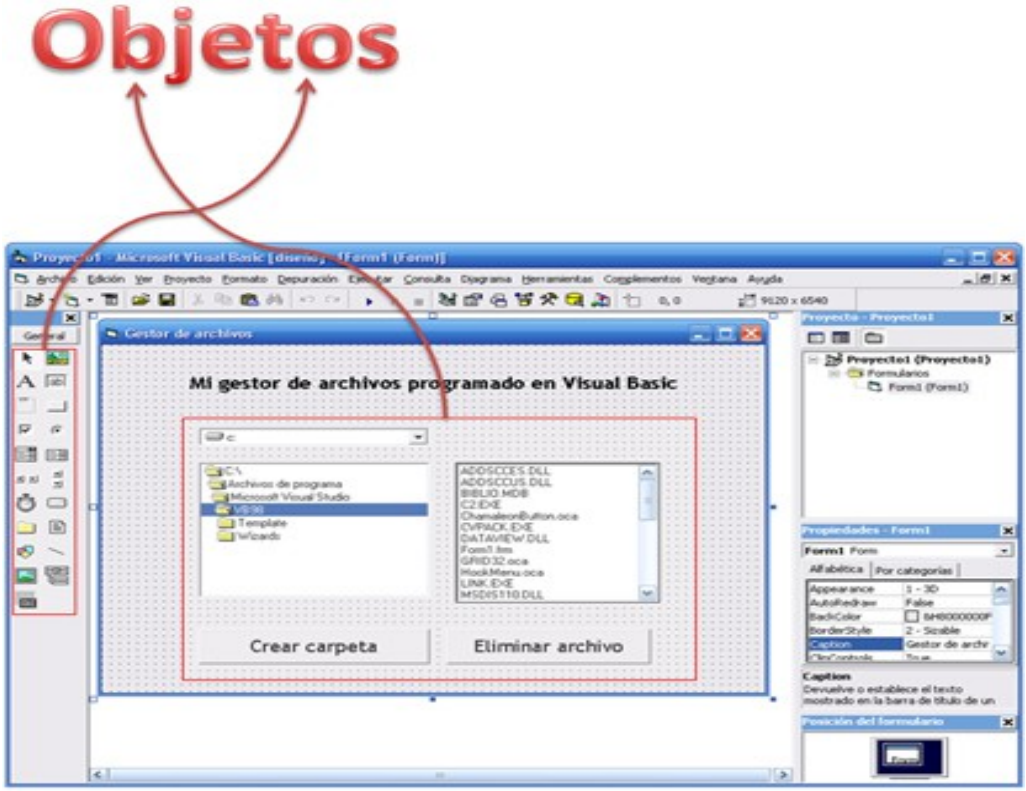
La mayor parte del código en un programa para un GUI está en los llamados **manejadores de eventos**. Cada manejador se encarga de realizar el conjunto de acciones asociadas a un evento determinado. Existe un **gestor de eventos** que se encarga de recibir todos los eventos de la aplicación y llamar al manejador adecuado.



La programación en GUIs suele ser un proceso iterativo de tres pasos:

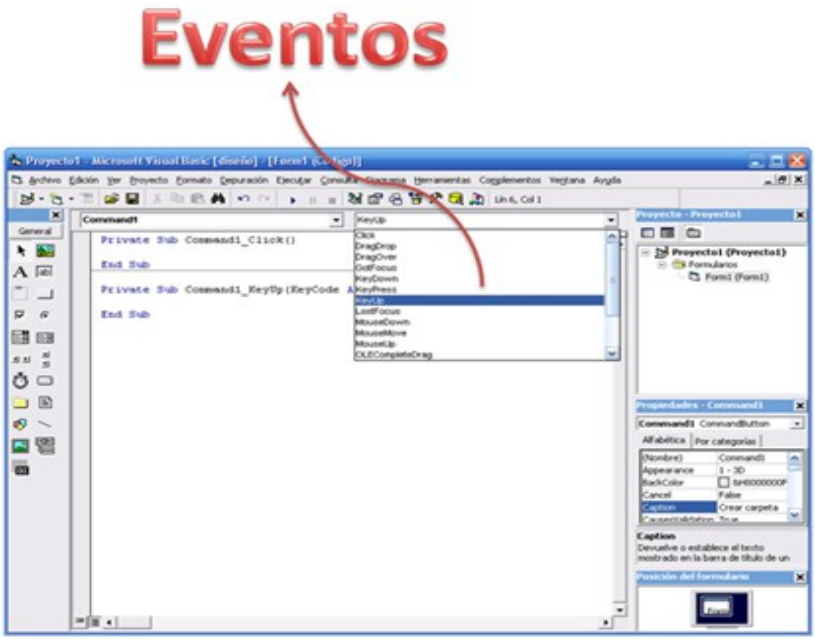
- **Diseñar la interfaz** de una parte de la aplicación, utilizando los widgets disponibles en el toolkit o barra de herramientas de desarrollo e incluye a su vez dos tareas:
  - Posicionar los widgets y establecer sus dimensiones
  - Modificar sus características visuales y funcionales (títulos, colores, comportamiento)
- Realizar la **captura de los eventos** de la interfaz que permitan implementar la funcionalidad requerida
- I**mplementar o codificar** cada uno de los manejadores correspondientes a los eventos capturados

Los IDES actuales pemiten realizar los dos primeros pasos de forma rápida.



Una vez realizado el diseño, los entornos de desarrollo eligen una de las siguientes estrategias para llevar a cabo la codificación de los eventos:

- Salvar el diseño y las propiedades en ficheros ocultos especiales que **son compilados junto al código** (Delphi, C++ Builder, Visual Basic)



- Generar el código correspondiente de la interfaz que el desarrollador completa con su propio código (NetBeans)

Como acabamos de ver, en un programa para un GUI la tarea fundamental a realizar es gestionar adecuadamente los eventos recibidos.

La información asociada a un evento suele ser como mínimo **un campo indicador del tipo de evento y el identificador del elemento** que genera dicho evento (botón, ventana, etc.)

Un aspecto fundamental en el diseño de un toolkit o IDE es la forma en que se produce la **conexión entre el gestor de eventos y los manejadores.**

En lenguajes primitivos como C el programador realiza tanto la implementación del gestor de eventos sino la llamada a los distintos manejadores.

En lenguajes más moderno el gestor de eventos ya está integrado en el propio IDE y no es necesario implementarlo.

6. Diálogos modales y no modales.

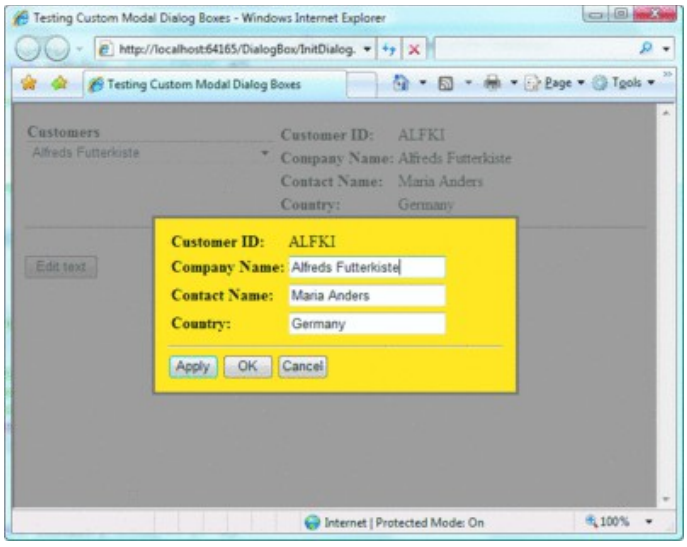
Las aplicaciones independientes tienen normalmente una **ventana principal,** que tanto muestra los datos principales sobre los que funciona la aplicación como expone la funcionalidad de procesamiento de datos a través de mecanismos de interfaz de usuario (UI) tales como barras de menús, barras de herramientas y barras de estado.

Una aplicación no trivial también puede mostrar ventanas adicionales para hacer lo siguiente:

- Mostrar información específica a los usuarios
- Recopilar información de los usuarios.
- Tanto mostrar como recopilar información.

Estos tipos de ventanas se conocen como ***cuadros de diálogo*** y hay dos tipos: **modales y no modales.**

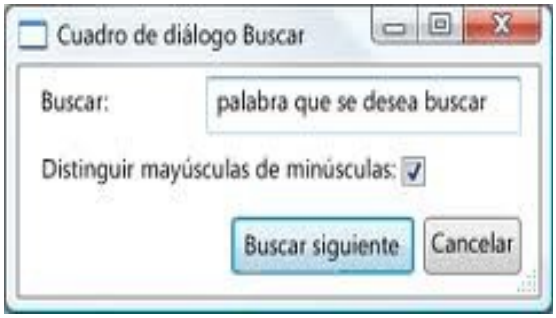
- Los cuadros de **diálogo *modales*** los muestra las funciones **cuando necesitan datos adicionales de los usuarios** para continuar. Dado que la función depende del cuadro de diálogo modal para recopilar los datos, el cuadro de diálogo modal también impide que un usuario active otras ventanas de la aplicación mientras permanece abierto. En la mayoría de los casos, los cuadros de diálogo modales permiten a los usuarios señalar que han terminado con el cuadro de diálogo modal presionando un botón **Aceptar** o **Cancelar**. Al presionar el botón **Aceptar** se indica que el usuario ha introducido los datos y desea que la función continúe su proceso con esos datos. Presionar el botón **Cancelar** indica que el usuario desea detener la ejecución de la función. Los ejemplos más comunes de cuadros de diálogo modales se muestran para abrir, guardar e imprimir datos.



Cuadro diálogo modal

- Un cuadro de **diálogo *no modal*,** por otra parte, **no impide que el usuario active otras ventanas mientras está abierto**. Por ejemplo, si un usuario desea buscar apariciones de una palabra determinada en un documento, una ventana principal abrirá habitualmente un cuadro de diálogo para preguntar al usuario qué palabra está buscando. Dado que la búsqueda de una palabra no impide que un usuario edite el documento, no obstante, no es necesario que el cuadro de diálogo sea modal. Un cuadro de diálogo no modal proporciona

al menos un botón **Cerrar** para cerrar el cuadro de diálogo y puede proporcionar botones adicionales para ejecutar funciones concretas, como un botón **Buscar siguiente** para buscar la palabra siguiente que coincida con los criterios de una búsqueda de palabra.

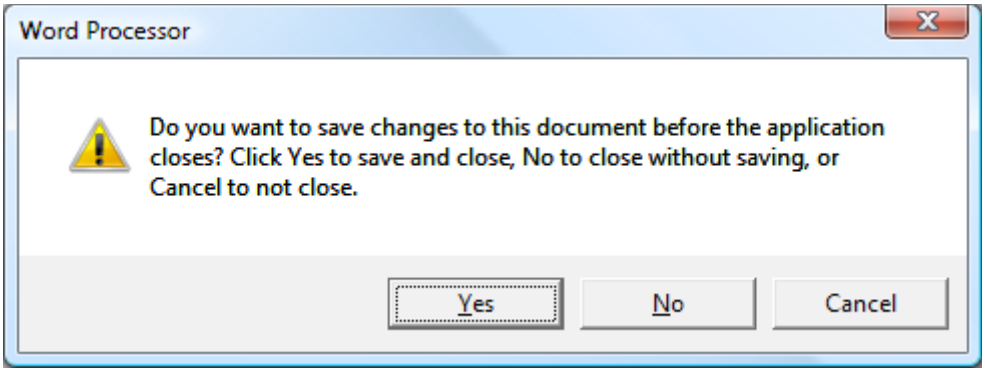


Cuadro diálogo no-modal

Entre los diferentes tipos de cuadros de diálogo los más comunes entre los diferentes IDE de programación nos podemos encontrar:

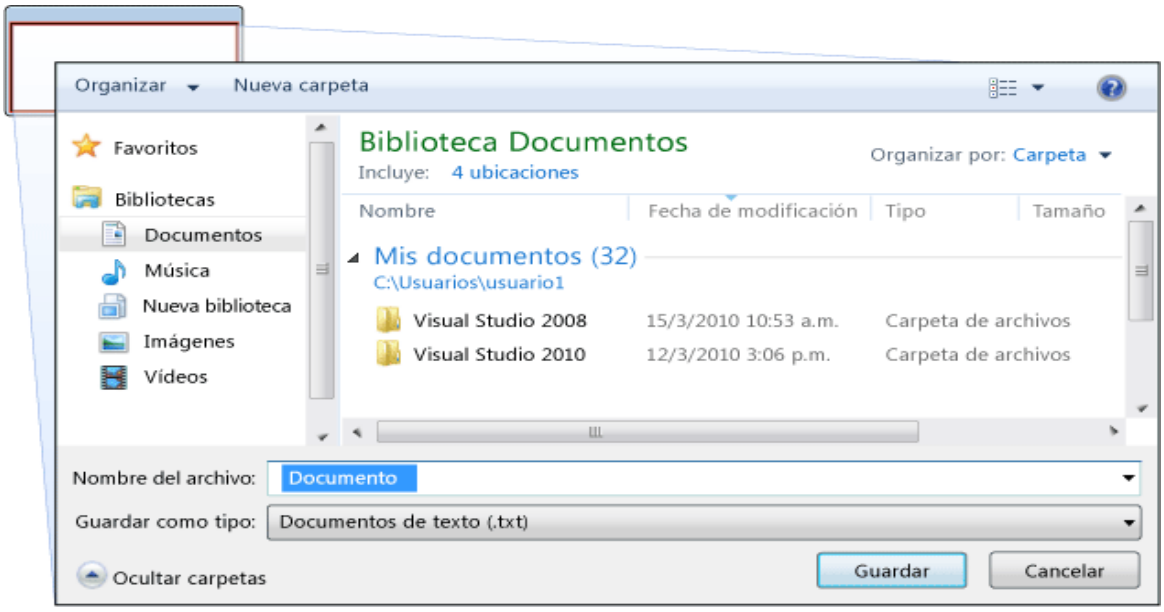
- **Cuadros de mensaje**

  Un *cuadro de mensaje* es un cuadro de diálogo que se puede utilizar para mostrar información textual y permitirles que los usuarios tomen decisiones con botones. La figura siguiente muestra un cuadro de mensaje que muestra información textual, hace una pregunta y proporciona al usuario tres botones para responder a la pregunta.



- **Cuadros de diálogo comunes**

  Aquí podemos encontrar la mayoría de los cuadros de diálogo más comunes y reutilizables por la mayoría de los IDE: *Imprimir, Abrir fichero, Guardar Fichero, …..*



- **Cuadros de diálogo personalizados**

  A veces se hace necesario crear nuevos cuadros de diálogo debido a que los más comunes antes citados no cumplen las expectativas del programador.

Para finalizar señalar que un abuso del uso de cuadros de diálogo, preferentemente modales, d**an lugar a diseños confusos para los usuarios de la aplicación.** En lo posible no deben llegarse a un tercer nivel de cuadros de diálogo, es decir, que desde la ventana principal de la aplicación no es aconsejable abrir más allá de tres cuadros de diálogo en modo jerárquico.

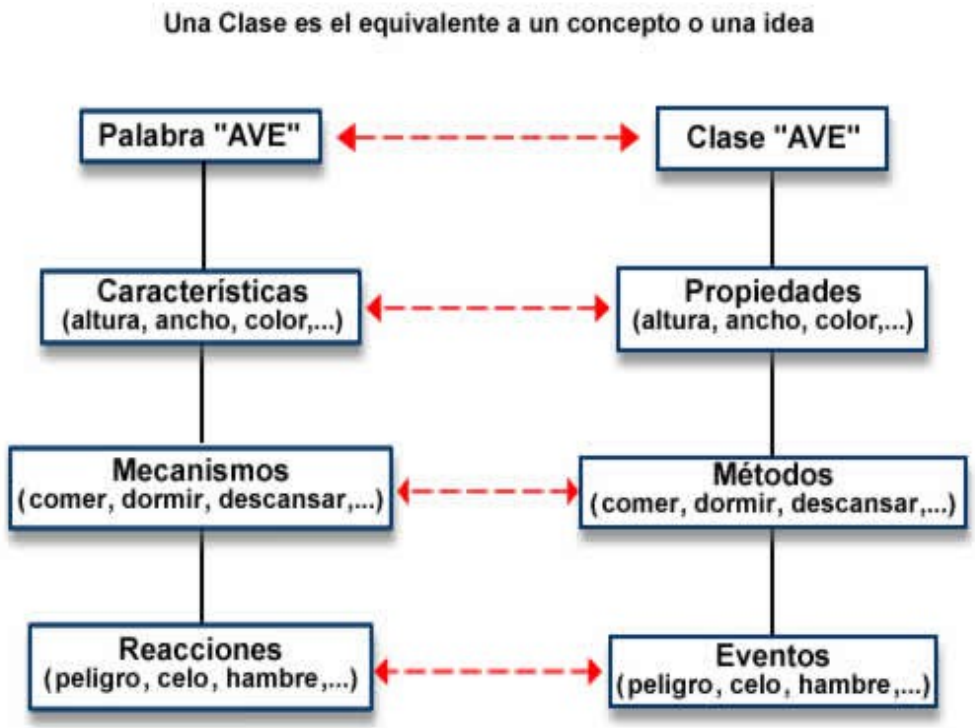
➔ Clases, propiedades y métodos

Una **clase** es una construcción que se utiliza como un modelo (o plantilla) para crear objetos de ese tipo iguales o con pequeñas variaciones. El modelo **describe el estado y el comportamiento** que todos los objetos de la clase comparten. Su comportamiento, **métodos,** y su estado, **atributos** se encapsulan**.**

Un **objeto** de una determinada clase se denomina una **instancia** de la clase. Una clase tiene tanto una interfaz y una estructura. La interfaz describe cómo interactuar con la clase y sus instancias con métodos, mientras que la estructura describe cómo los datos se dividen en atributos dentro de una instancia.

La funcionalidad de una clase se implemente mediante los **métodos.** Cuando se desea realizar una acción sobre un objeto se desencadenará un **evento**, se dice que **se le manda un mensaje** invocando a un método que realizará la acción.

Las **propiedades** son un tipo especial de métodos. Debido a que suele ser común que las variables miembro sean privadas para **controlar el acceso** y mantener la coherencia, surge la necesidad de permitir consultar o modificar su valor mediante pares de métodos: *GetVariable y SetVariable.*



Más técnicamente, una clase es un conjunto coherente que consiste en un tipo particular de **metadatos.** Los **metadatos** han cobrado gran relevancia en el mundo de Internet, por la necesidad de utilizar los metadatos para la clasificación de la enorme cantidad de datos. Además de la clasificación los metadatos pueden ayudar en las búsquedas. Por ejemplo, si buscamos un articulo sobre vehículos, este dato tendrá sus correspondiente metadatos clave adjuntos, como 4 ruedas, cuatro ruedas, motor, etc. Hoy en día, por ejemplo, es común codificar datos mediante **XML** así son legibles tanto para humanos como para computadores.

➔ Eventos: escuchadores.

Una vez se han “dibujado” la interfaz gráfica en la pantalla la aplicación suele quedar a la espera sin ejecutar código. Cuando se acciona sobre la interfaz (click botón, escritura....) el escuchador correspondiente a uno de los componenets gráficos se activa.

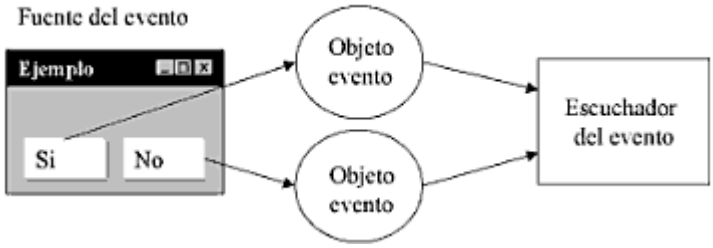
public interface IListener

{

public void Update(String eventType, Object oldValue, Object newValue);

}

En JAVA las clases escuchadoras de eventos se denominan o llevan el término *Listener.*



Suele ser en los escuchadores donde el programador incorpora el código específico para responder a los eventos generados por el usuario u otra instancia de la aplicación.

Como se ve en la figura los escuchadores son objetos que pueden recibir y manejar eventos enviados por otros objetos y para ello un escuchador debe:

- implementar las interfaces escuchadores de eventos
- registrarse en la fuente de eventos

Algunos de los escuchadores de java con respecto a sus objetos

INTERFAZ	MÉTODOS
ActionListener	actionPerformed(ActionEvent)
FocusListener	focusGained(FocusEvent) focusLost(FocusEvent)
TextListener	textValueChanged(TextEvent)
KeyListener	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)
MouseListener	mouseClicked(MouseEvent) mousePressed(MouseEvent) mouseReleased(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent)
MouseMotionListener	mouseDragged(MouseEvent) mouseMoved(MouseEvent)
WindowListener	windowActivated(WindowEvent) windowClosed(WindowEvent) windowClosing(WindowEvent) windowDeactivated(WindowEvent) windowDeiconified(WindowEvent) windowIconified(WindowEvent) windowOpened(WindowEvent)

Cuando un objeto genera un evento (fuente) realiza una llamada a determinados métodos definidos por el escuchador registrado en la fuente. Antes de iniciar la gestión del evento, la fuente precisa conocer si el escuchador ha implementado los métodos a llamar y eso se realiza mediante **la interfaz** del escuchador de eventos. Por ello el escuchador debe implementar los métodos de la interfaz del escuchador de aquellos eventos que esté interesado en escuchar.

La interfaz del escuchador tienen tantos métodos implementados como eventos puede manejar.