

# TEMA 1

INTRODUCCIÓN AL DI,  
USABILIDAD, ACCESIBILIDAD  
Y  
PROTOTIPADO

# ÍNDICE

<b>INTRODUCCIÓN.....</b>	<b>3</b>
<b>EVOLUCIÓN HISTÓRICA DE LAS INTERFACES .....</b>	<b>3</b>
<i>Xerox Alto (1973).....</i>	<i>3</i>
<i>Xerox Star (1981).....</i>	<i>5</i>
<i>Apple Lisa (1983).....</i>	<i>6</i>
<i>Proyecto Athena (1984).....</i>	<i>6</i>
<i>Workbench (1985).....</i>	<i>7</i>
<i>Microsoft Windows 1.0 (1985).....</i>	<i>8</i>
<i>NeXTSTEP (1985).....</i>	<i>8</i>
<i>GNOME (1985).....</i>	<i>10</i>
<i>Windows 3.0 (1990).....</i>	<i>11</i>
<i>Otros interfaces.....</i>	<i>11</i>
<b>TIPOS DE INTERFACES.....</b>	<b>12</b>
CLI (COMMAND LÍNEA INTERFACE) .....	12
GUI (GRAPHICAL USER INTERFACE) .....	13
NUI (NATURAL USER INTERFACE).....	13
INTERFACES DEL FUTURO OUI (INTERFAZ DE USUARIO ORGÁNICA).....	14
<b>MODELADO DE UNA INTERFAZ DE USUARIO .....</b>	<b>16</b>
MODELO DEL USUARIO .....	16
MODELO DEL PROGRAMADOR.....	16
MODELO DEL DISEÑADOR.....	16
<b>USABILIDAD .....</b>	<b>18</b>
PRINCIPIOS .....	18
DISTRIBUCIÓN DE LOS ELEMENTOS EN PANTALLA .....	24
VALIDACIÓN.....	25
INTERNACIONALIZACIÓN.....	27
<i>Internacionalización en JavaFX .....</i>	<i>29</i>
TRABAJO EN SEGUNDO PLANO.....	31
GUÍAS DE DISEÑO (RIBBON, LOLIPOP, ETC.) .....	33
ENLACES DE INTERÉS .....	34
<b>ACCESIBILIDAD .....</b>	<b>36</b>
CONCEPTO .....	36
ACCESIBILIDAD .....	36
<b>PROTOTIPADO .....</b>	<b>38</b>

# Introducción

El diseño de interfaces de usuario es una tarea que ha adquirido relevancia en el desarrollo de un sistema. **La calidad de la interfaz de usuario puede ser uno de los motivos que conduzca a un sistema al éxito o al fracaso.**

En la actualidad el contacto con el ordenador, el móvil, la tablet,... se ha vuelto una actividad muy común y necesaria, es por tal motivo que ha surgido la necesidad de **ambientes amigables que faciliten el uso de todas las herramientas de estos dispositivos**. Es así como se han creado las llamadas Interfaces Gráficas de Usuario (en inglés Graphic User Interface GUI).

De hecho, es **gracias a la interfaz gráfica de usuario que los sistemas operativos se convirtieron en soluciones mucho más visuales y accesibles para el público**, algo que hasta la fecha era complicado debido a las funciones de unos sistemas basados en modo texto que resultaban demasiado limitados para un gran número de escenarios.

## Evolución histórica de las interfaces

Lewis y Rieman (1993) definen:

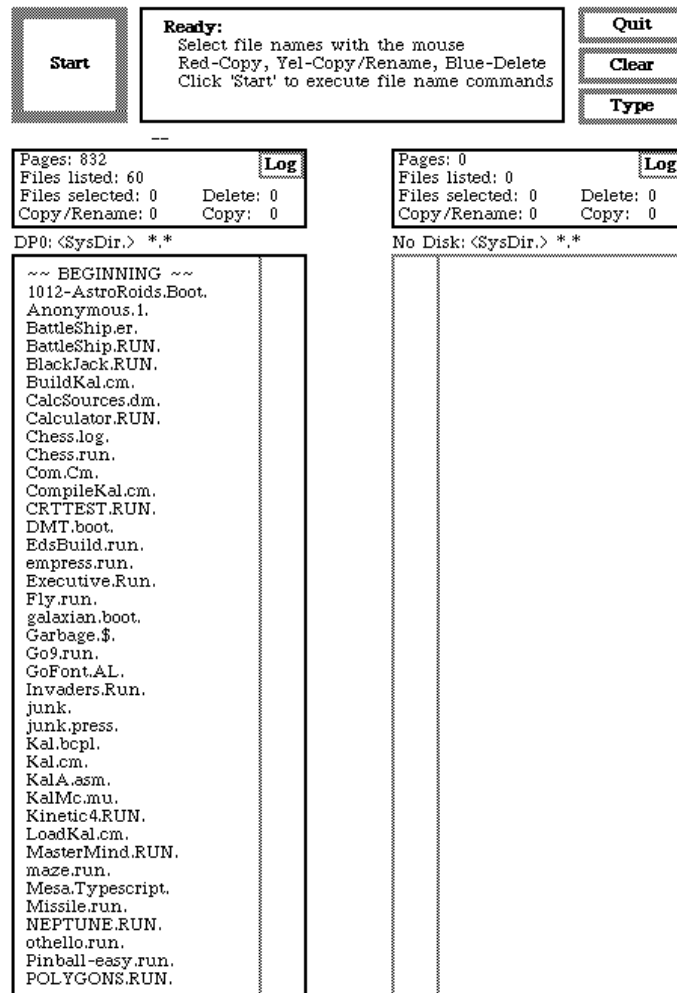
Las interfaces básicas de usuario son aquellas que incluyen cosas como menús, ventanas, teclado, ratón, los beeps y algunos otros sonidos que la computadora hace, en general, todos aquellos canales por los cuales se permite la comunicación entre el hombre y la computadora

Dicho de otra forma, **la interfaz es la mediadora entre dos sistemas de diferente naturaleza: el hombre y la máquina**; ya que además de facilitar la comunicación y la interacción entre ambos, sirve de traductor, pues estos dos sistemas “hablan” lenguajes diferentes

Xerox Alto (1973)

En 1973, Xerox PARC desarrolló la computadora personal, Alto. Tenía una pantalla de mapa de bits y **fue la primera computadora en demostrar la metáfora de escritorio y la interfaz gráfica de usuario** (GUI, graphical user interface). No fue un producto comercial, pero varios miles de unidades fueron construidas y fueron ampliamente utilizadas en PARC, al igual que en otras oficinas de Xerox y en varias universidades durante años. El Alto influyó en gran medida el diseño de las computadoras personales durante el final de los años 70 y el principio de los años 80, notablemente el Three Rivers PERQ, los Apple Lisa y Macintosh y la primera estación de trabajo de Sun Microsystems.

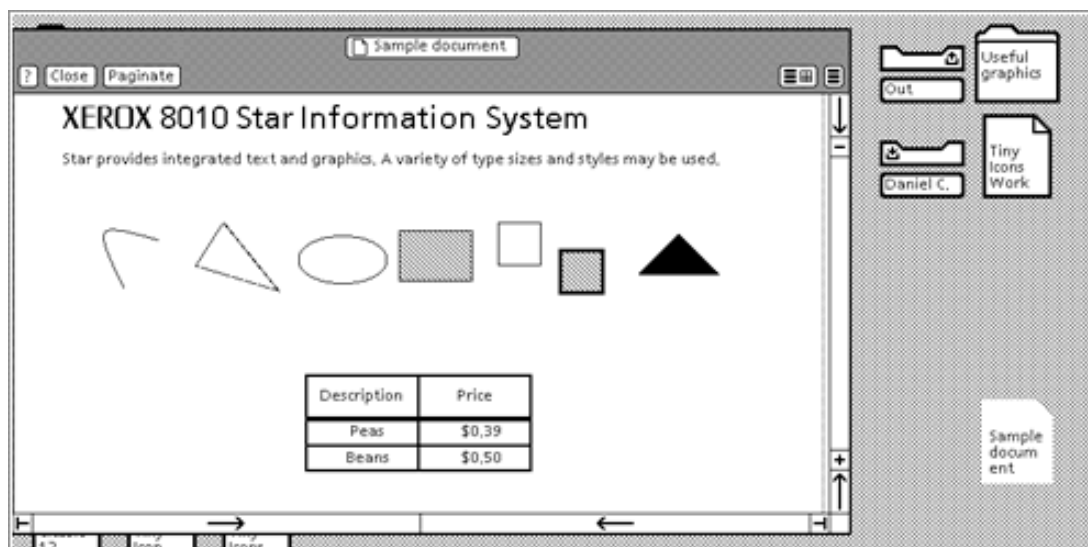
El GUI fue inicialmente desarrollado en Xerox PARC por Alan Kay, Larry Tesler, Dan Ingalls y algunos investigadores más. **Usaba ventanas, iconos y menús, incluyendo el primer menú desplegable fijo, para dar soporte a comandos como abrir ficheros, borrar y mover ficheros**, etc. En 1974 se comenzó en PARC a trabajar en Gypsy, el primer editor de texto gráfico WYSIWYG (What-You-See-Is-What-You-Get, “lo que ves es lo que consigues”). En 1975 los ingenieros de Xerox presentaron una demostración de un GUI “incluyendo iconos y el primer uso de menús emergentes”.



## Xerox Star (1981)

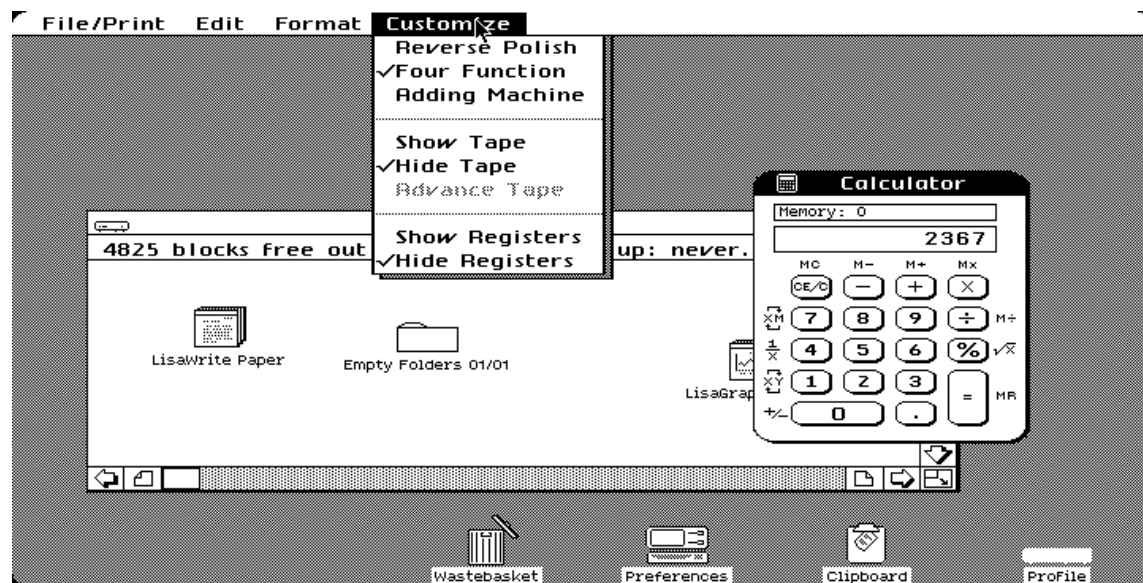
En 1981 Xerox presentó un producto innovador, el Star, incorporando muchas de las innovaciones de PARC. Xerox Star 8010 fue la primera estación de trabajo con interface gráfica y ratón incorporado.

**Aunque no fue un éxito comercial, el Star influyó de manera importante los futuros desarrollos, por ejemplo en Apple, Microsoft y Sun Microsystems.**



## Apple Lisa (1983)

Tras una visita al Xerox PARC en 1979, el equipo de Apple encabezado por Jef Raskin se concentra en diseñar un entorno gráfico para su nueva generación de 16 bits, que se verá plasmado en el Apple Lisa en 1983. Ese sistema gráfico es el sucesor del Apple II, el Apple II GS. Un segundo equipo trabaja en el **Apple Macintosh que verá la luz en 1984 con una versión mejorada del entorno gráfico del Lisa** (pretendimos hacer un ordenador tan simple de manejar como una tostadora). Desde ese momento el Mac reinará como paradigma de usabilidad de un entorno gráfico; pese a que por debajo el sistema operativo sufra cambios radicales, los usuarios no avanzados no son conscientes de ello y no sufren los problemas de otras plataformas.



## Proyecto Athena (1984)

Fue desarrollado en el MIT por Bob Scheifler y Jim Gettys en el proyecto Athena. Funciona sobre el sistema operativo Unix.

**El ordenador en el cual se visualizan los gráficos actúa como servidor mientras que el programa que los genera se ejecuta como cliente.**

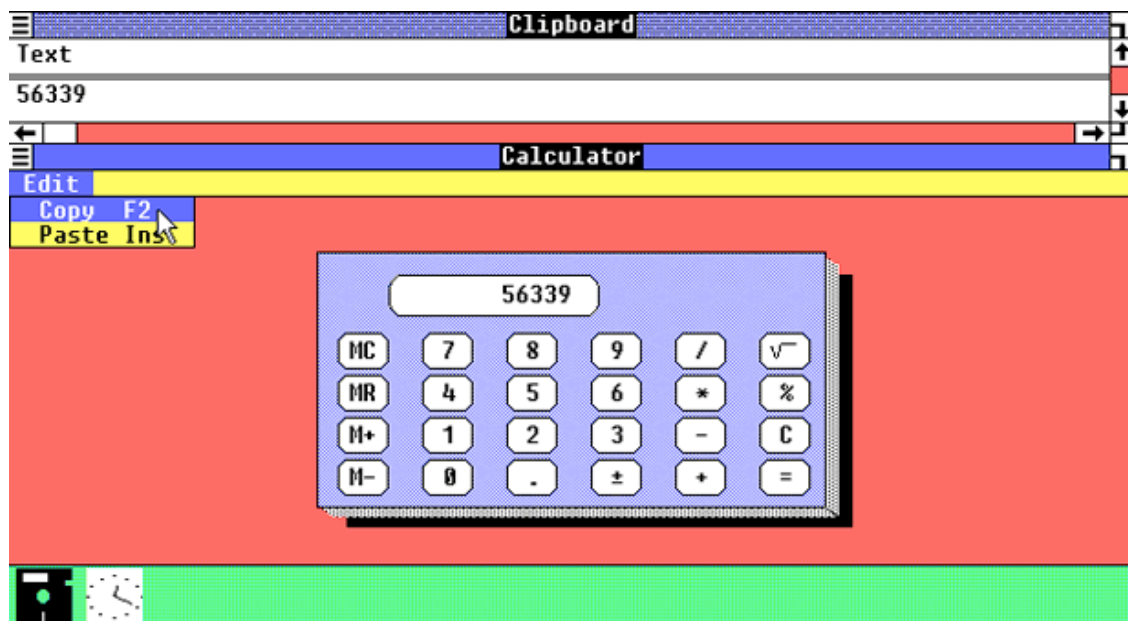
**X Windows** system 1 sigue utilizándose para todos los sistemas Linux.





Microsoft Windows 1.0 (1985)

**Windows 1.0, una GUI para MS-DOS, fue lanzado en 1985.** La aparición de Windows 1.0 es un paso adelante en la forma en que máquina y persona interactúan. La respuesta del mercado no fue muy entusiasta y **no consigue gran aceptación hasta que, llegado 1990, Microsoft lanza Windows 3.0.**



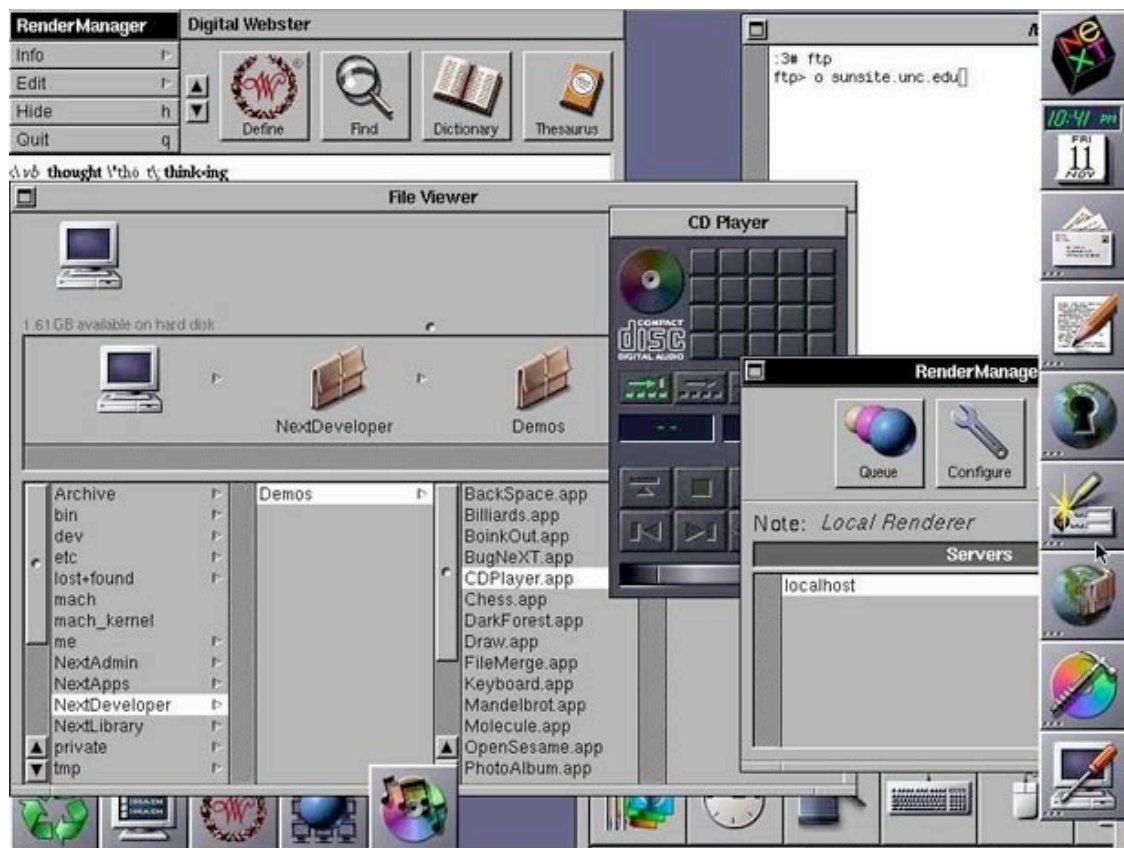
NeXTSTEP (1985)



**Realizado por NeXT compañía fundada por Steve Jobs** en su “travesía del desierto”, es decir **cuando estuvo exiliado de Apple Computer**.

**NeXTstep** dio paso al OPENSTEP, una versión más abierta que **buscaba ser independiente de los dispositivos**, Apple lo compró y lo integró en lo que acabaría siendo el desarrollo de Rhapsody, desembocando éste último en lo que hoy conocemos como Mac OSX.

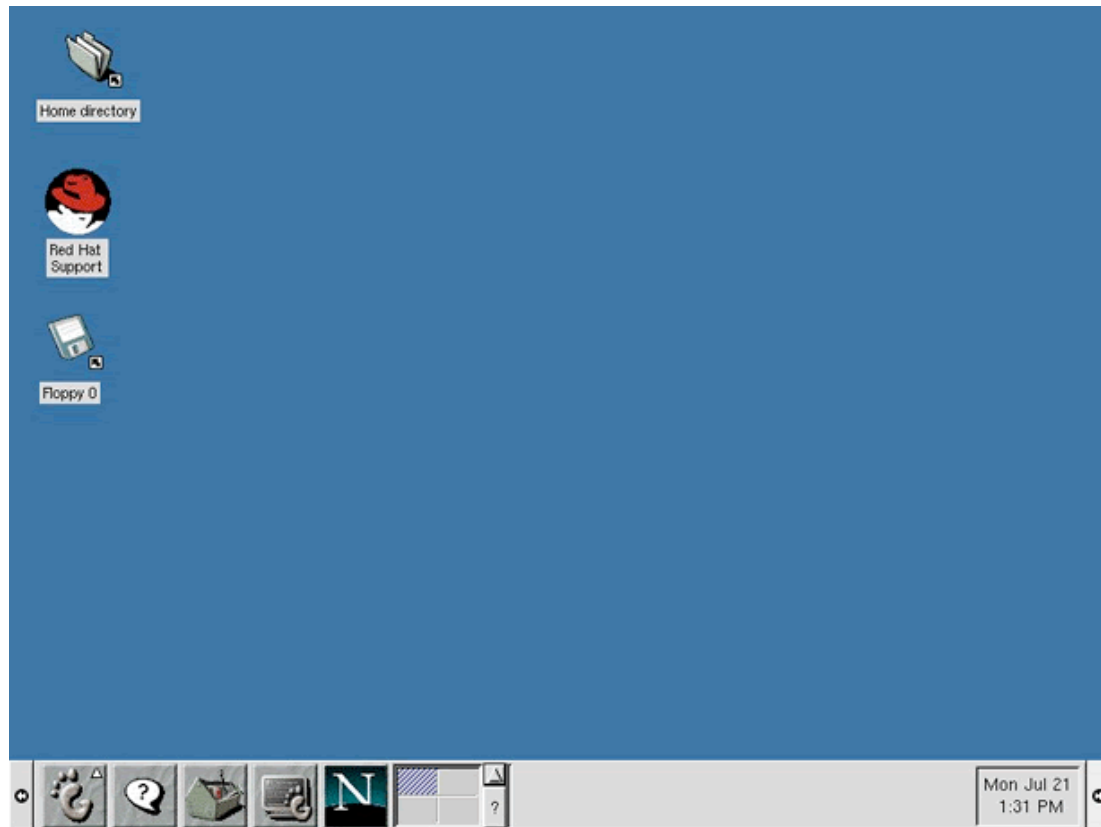
La GUI de NeXTStep fue **la primera en incluir el arrastre completo de las ventanas** en su interfaz de usuario, **en lugar de solo mostrar un recuadro vacío** durante el arrastre, en una máquina comparativamente poco potente para lo normal actualmente, idealmente ayudada por hardware gráfico de altas prestaciones.



## GNOME (1985)

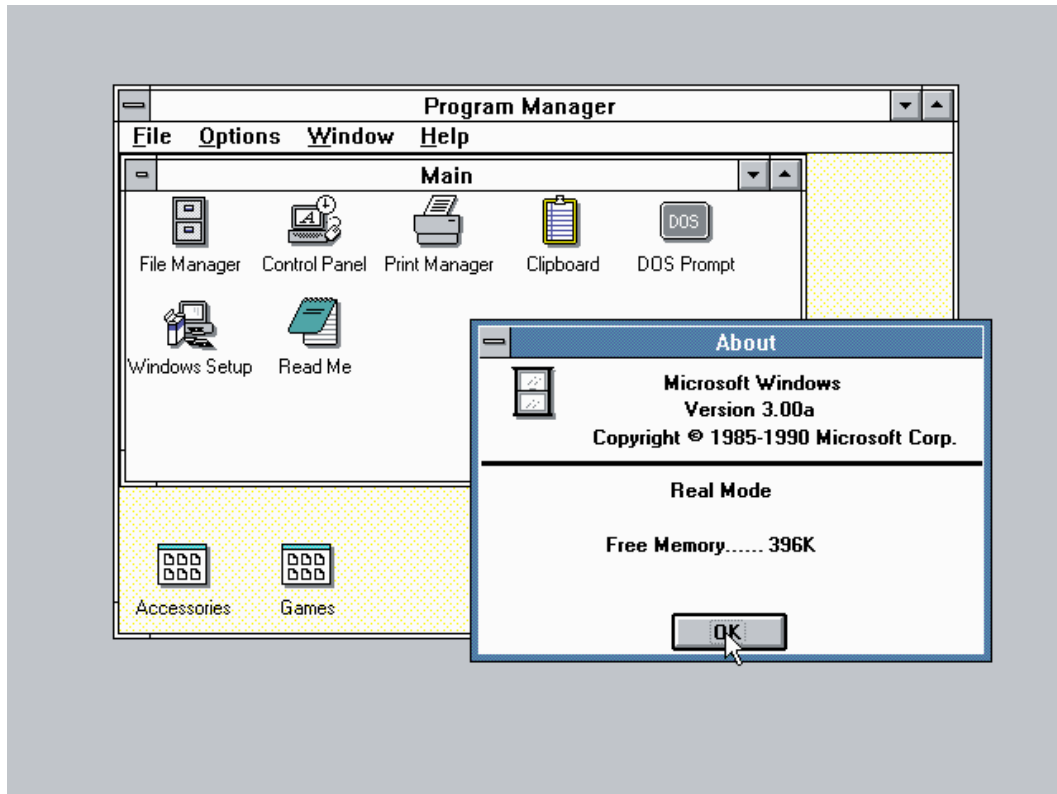
Gnome fue un esfuerzo internacional por construir una **interface gráfica de usuario utilizando exclusivamente software libre**.

**Es parte de GNU Project** y puede ser usado con varios sistemas operativos Unix, la mayoría de las librerías de Linux, y como parte del Java Desktop System en Solaris. El proyecto fue iniciado por los mexicanos Miguel de Icaza y Federico Mena



## Windows 3.0 (1990)

No fue hasta 1990 con el lanzamiento de Windows 3.0, basado en Common User Access que la popularidad del sistema verdaderamente aumento. El sistema operativo admite modos mejorados estándar y 386, que permitieron el uso de la mayor capacidad de memoria de 640 KB y espacio en disco duro, lo que resulta en la capacidad de utilizar mayores resoluciones de pantalla y mejores gráficos, tales como Super VGA 800 × 600 y 1024 × 768.



## Otros interfaces

OS/2 1.x (1988), Workbench 2.04 (1991), Mac OS System 7 (1991), OS/2 Warp 4 (1996), BeOS 4.5 (1999),...

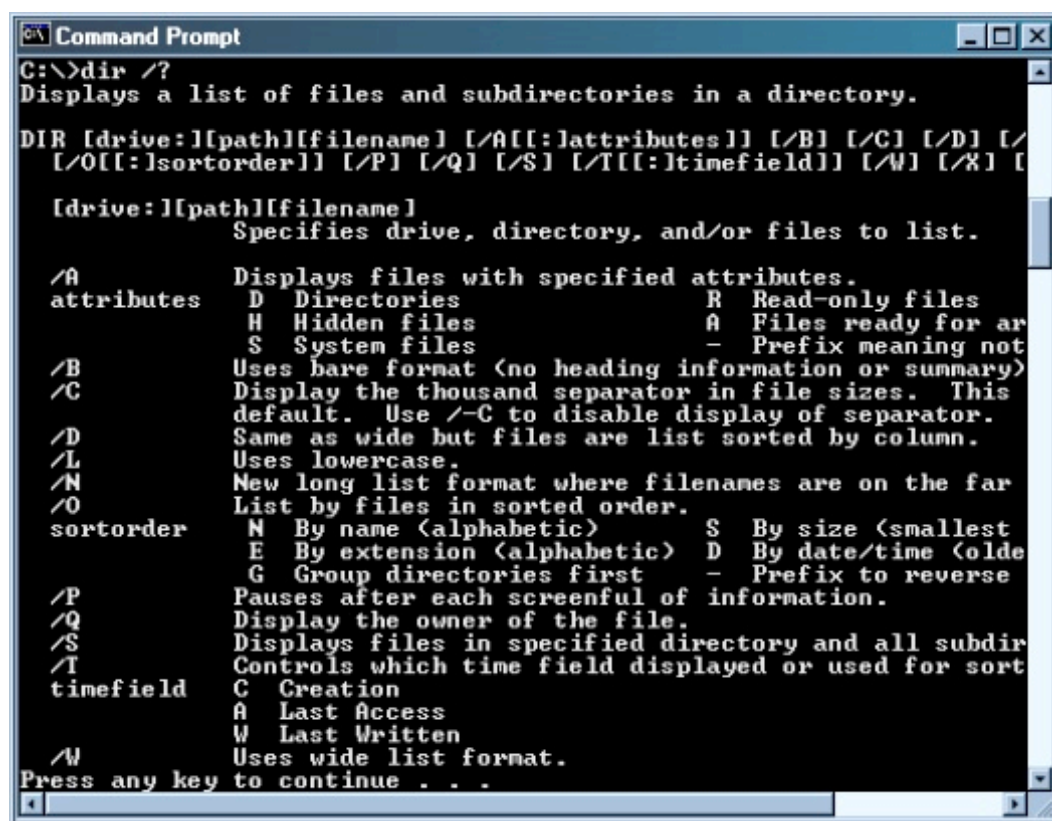
# Tipos de interfaces

En función de cómo interactuemos con el ordenador, podemos clasificar las interfaces de usuario de diferente manera.

## CLI (Command línea interface)

Es una interfaz de usuario en la que una persona interactúa con la información digital a través de un entorno textual y ordenes escritas por el usuario por medio de un interfaz físico del tipo teclado.

Este tipo de interfaces se caracterizan en que el usuario no recibe mucha información por parte del sistema. Debe conocer cómo funciona el ordenador.



```
Command Prompt
C:\>dir /?
Displays a list of files and subdirectories in a directory.

DIR [drive:][path][filename] [/A[:attributes]] [/B] [/C] [/D] [/O[:sortorder]] [/P] [/Q] [/S] [/T[:timefield]] [/W] [/X] [
[drive:][path][filename]
    Specifies drive, directory, and/or files to list.

/A      Displays files with specified attributes.
attributes  D Directories                R Read-only files
              H Hidden files              A Files ready for archiving
              S System files              - Prefix meaning not to search
/B      Uses bare format (no heading information or summary)
/C      Display the thousand separator in file sizes. This
        default. Use /-C to disable display of separator.
/D      Same as wide but files are list sorted by column.
/L      Uses lowercase.
/N      New long list format where filenames are on the far
/O      List by files in sorted order.
sortorder  N By name (alphabetic)          S By size (smallest to largest)
              E By extension (alphabetic)  D By date/time (oldest to newest)
              G Group directories first    - Prefix to reverse order
/P      Pauses after each screenful of information.
/Q      Display the owner of the file.
/S      Displays files in specified directory and all subdirectories.
/T      Controls which time field displayed or used for sort
timefield  C Creation
              A Last Access
              W Last Written
/W      Uses wide list format.
Press any key to continue . . .
```

### Ventajas

- Potente, flexible y controlado por el usuario. Mejor para usuarios experimentados.
- Para los usuarios expertos es más rápido, de vez en cuando se incluye un CLI como parte de un interfaz, que se utiliza cuando el usuario tiene más experiencia.

### Inconvenientes

- Carga de memoria del usuario
- Nombres no siempre adecuados en los mandatos (del, erase)
- Inflexible en la sintaxis

Algunos ejemplos de este tipo de interfaces son el Shell de GNU/Linux, el Command de los SO Windows.

## GUI (Graphical User Interface)

Es una interfaz de usuario en la que una **persona interactúa con la información digital a través de un entorno gráfico de simulación.**

Algunos ejemplos de este tipo de interfaces son el escritorio de Windows, GNOME y Unity en Linux



## NUI (Natural User Interface)

También conocidas como interfaces TUI (Tangible User Interface), es una interfaz de usuario en el que una **persona a interactúa con la información digital a través del medio físico.**

Algunos ejemplos de este tipo de interfaces son el reconocimiento de voz y movimientos, seguimiento de mirada,...



Una comparativa entre estos tipos de interfaces nos lleva a las siguientes conclusiones



## Interfaces del futuro OUI (Interfaz de usuario orgánica)

Las OUI son **aquellas interfaces que son capaces de modificar y adaptar su forma**. A estas interfaces se las conoce como las interfaces del futuro y permitirán cambiar la forma en la que nos relacionamos con los ordenadores.

## CLI

*Command Line  
Interface  
(Texto)*



## GUI

*Graphical User  
Interface  
(Gráficos)*



## NUI

*Natural user  
Interface  
(Objetos)*



## OUI

*Organic User  
Interface  
(Orgánico)*



- Recordar

- Reconocimiento

- Intuición

- Síntesis

- Estático

- Receptivo

- Evocativo

- Fluido

- Directo

- Exploratorio

- Contextual

- Anticipativo

- Desconectado  
(Abstracto)

- Indirecto

- Sin mediación  
(Directo)

- Extensivo

- Alto-Bajo

- Doble Medio

- Rápido Poco

- Cero constante

# Modelado de una interfaz de usuario

Existen tres puntos de vista:

1. Usuario
2. Programador
3. Diseñador

Cada uno tiene su propio esquema mental de la interfaz, que obtiene a través de su experiencia.

## Modelo del usuario

El usuario tiene su visión personal del sistema, y espera que éste se comporte de una cierta forma. Se puede conocer el modelo del usuario estudiándolo, ya sea realizando tests de usabilidad, entrevistas, o a través de una retroalimentación. **Son de gran utilidad las metáforas que asocian una acción nueva a una ya conocida por el usuario.** Un ejemplo típico de metáfora es el escritorio

## Modelo del programador

Es el más fácil de visualizar, al poderse especificar formalmente. Está **constituido por los objetos que manipula el programador**, distintos de los que trata el usuario, por ejemplo, **el programador llama base de datos a lo que el usuario llama agenda**. El programador conoce la plataforma de desarrollo, el sistema operativo, las herramientas de desarrollo, las especificaciones.

## Modelo del diseñador

**El diseñador mezcla las necesidades, ideas, deseos del usuario y los materiales de que dispone el programador para diseñar un producto de software.** Actúa de intermediario entre ambos.

**El modelo del diseñador** describe los objetos que utiliza el usuario, su presentación al mismo y las técnicas de interacción para su manipulación. **Consta de 3 partes:**

Presentación. Es **lo primero que capta la atención del usuario, pero más tarde pasa a segundo plano** adquiriendo más importancia la interacción con el producto

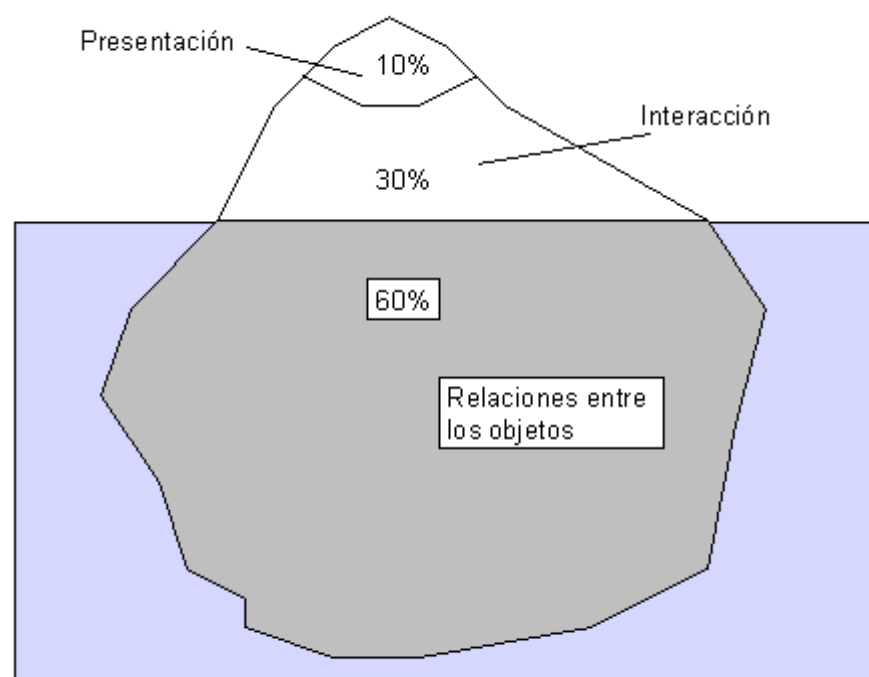
Interacción. Define **las técnicas de interacción del usuario, a través de diversos dispositivos**. Adquiere mucha importancia para usuarios con discapacidades

Relaciones entre los objetos. **Es la más importante y es donde el diseñador determina la metáfora adecuada que encaja con el modelo mental del usuario.** El modelo debe



comenzar por esta parte e ir hacia arriba. Una vez definida la metáfora y los objetos de la interfaz, los aspectos visuales saldrán de una manera lógica y fácil.

Podríamos representar de manera gráfica estos tres elementos de la siguiente forma:



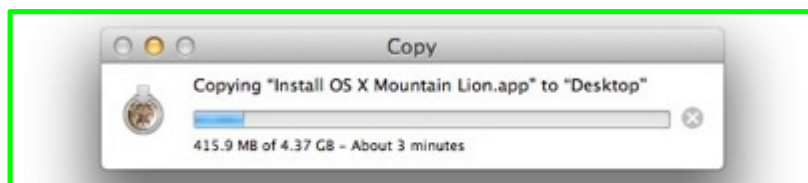
# Usabilidad

El diseño de interfaces usables ha pasado de ser una opción a una obligación en los últimos tiempos, empujado en gran parte por la necesidad de elegir muy bien lo que se muestra por pantalla en un dispositivo que tiene unas pocas pulgadas de tamaño (i.e. móviles, tablets y wereables). Hay mucha literatura al respecto, muchos patrones e interfaces gráficas en las que inspirarnos, y muchas guías que nos dicen cómo realizar un diseño centrado en el usuario. En las siguientes secciones estudiaremos las bases del UI (User Interface), UX (User eXperience) e IxD (Interaction Design)

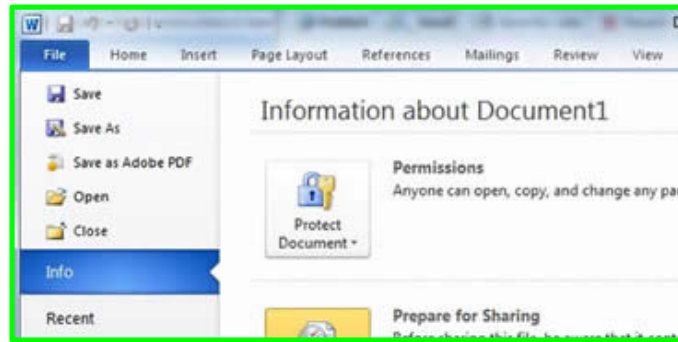
## Principios

Los diversos autores coinciden en fijar unos principios básicos que guían el diseño de las interfaces usables. Aunque cada uno tiene su propia visión y denominación de principio, la mayoría de ellos coinciden en los siguientes:

1. **Adoptar el punto de vista del usuario.** Para ello es necesario ver la interfaz desde fuera y en relación con las tareas que se han de realizar. El usuario debe poder trabajar con la herramienta sin necesidad de usar un manual, por lo que debe ser intuitivo. Para ello es necesario la implicación del usuario final en el desarrollo. Se recomienda para este punto realizar un **prototipo** de la aplicación en papel.
2. **Realimentación:** La interfaz debe dar inmediatamente alguna respuesta a cualquier acción del usuario. Este principio es vital, y muchas interfaces no lo cumplen. Ejemplo de ello son:
  - a. Barra de progreso: nos indica que la interfaz está trabajando, pero no se ha quedado colgada. Si no informamos de alguna manera al usuario, puede pensar que la aplicación no está haciendo nada. Mejor cuanto más información demos: no solo barra, sino también texto.

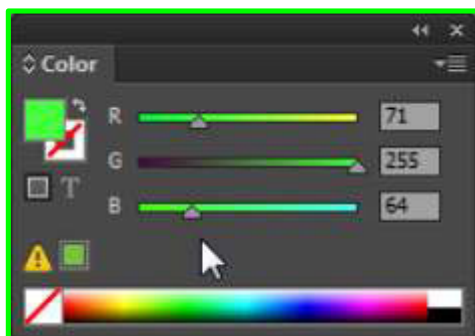


- b. Resaltar la opción elegida de menú. Si seleccionamos una opción de un menú y no cambia nada, el usuario no puede saber si se ha aplicado acción alguna.

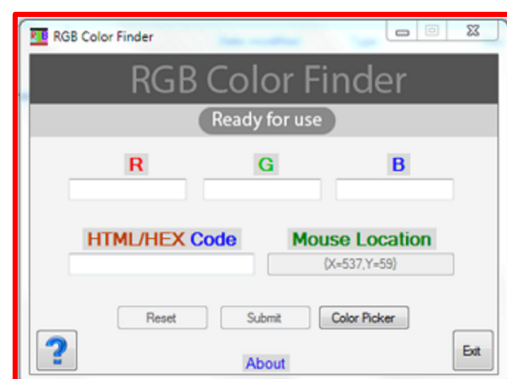


- c. Comunicar el éxito o fracaso de una operación. Por ejemplo, si en una tienda online realizamos una compra y al finalizar no nos informa de que hemos realizado la compra correctamente, el usuario puede llegar a pensar que ha ocurrido un error.
3. **Consistencia u homogeneidad.** La aplicación debe comportarse y ser visualmente similar en sus distintas pantallas o fases. Un buen ejemplo de esto son los menús, que siempre siguen el mismo orden: Archivo, Editar, Ver, etc. El objetivo es que el usuario pueda generalizar sus conocimientos de un aspecto de la interfaz a otro. Por ejemplo, en este documento se usa el **rojo** para malas interfaces, y el **verde** para buenas.
  4. **Tratamiento de errores:** Minimizar la posibilidad de cometer errores.
    - a. Usar preferentemente controles que restrinjan los valores a introducir. Por ejemplo usar un spinner para números en lugar de campo de texto.

ASÍ SÍ:

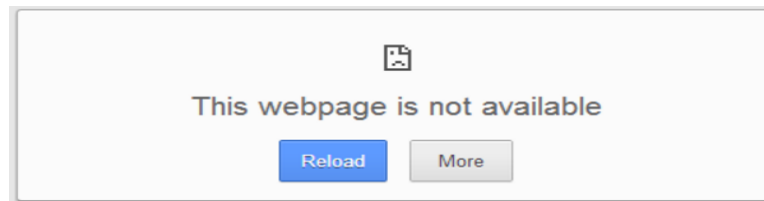


ASÍ NO:

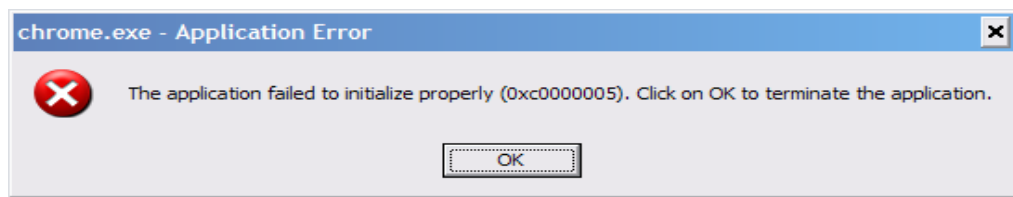


- b. Inhibir controles que representan acciones no permitidas. Por ejemplo, dejar habilitado un botón cuando no se puede pulsar.
- c. Pedir la confirmación de operaciones peligrosas.
- d. Evitar los controles que sean físicamente difíciles de manejar (controles muy pequeños, menús emergentes o submenús con muchas opciones).

- e. En caso de error, dar información suficiente y adaptada al punto de vista del usuario, tanto a sus objetivos como a sus conocimientos. Así sí:



Así NO:



- f. Facilitar la recuperación en caso de error, por ejemplo con Ctrl + Z.

5. **Minimizar la necesidad de memorización:** (véase libro “Don’t make me Think!”)

- a. Usar controles gráficos en lugar de campos de texto. Limitan el error y la necesidad de aprender a usarlos.
- b. Utilizar nombres y símbolos autoexplicativos y fáciles de recordar.

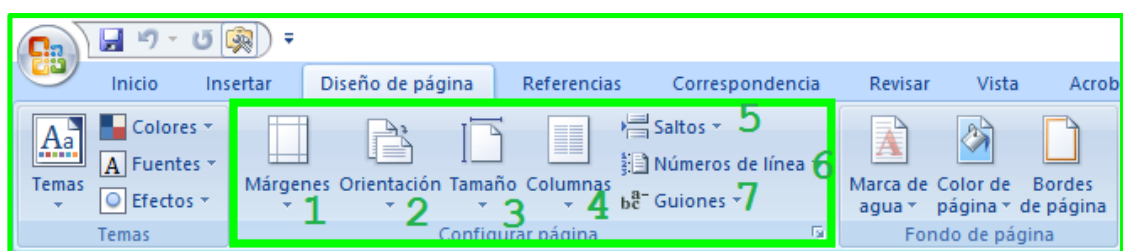
¿Esto es dentro o fuera?



Mucho mejor así

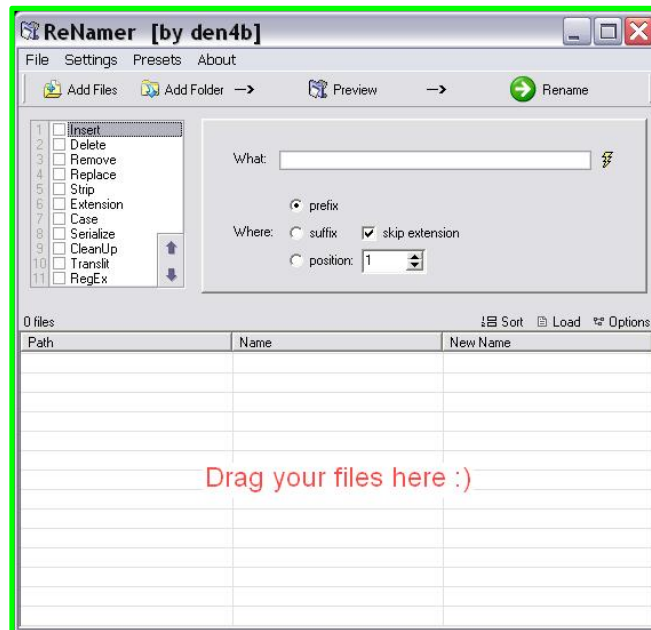


- c. **Minimalismo**, limitar la carga de información a corto plazo. Tener **sólo la información necesaria en pantalla** (reglas de “*menos es más*” y del  $7 \pm 2$ ). Por ejemplo la interfaz Ribbon de Microsoft agrupa los controles siguiendo esta regla. Las opciones siempre se muestran en grupos de, como máximo 7.

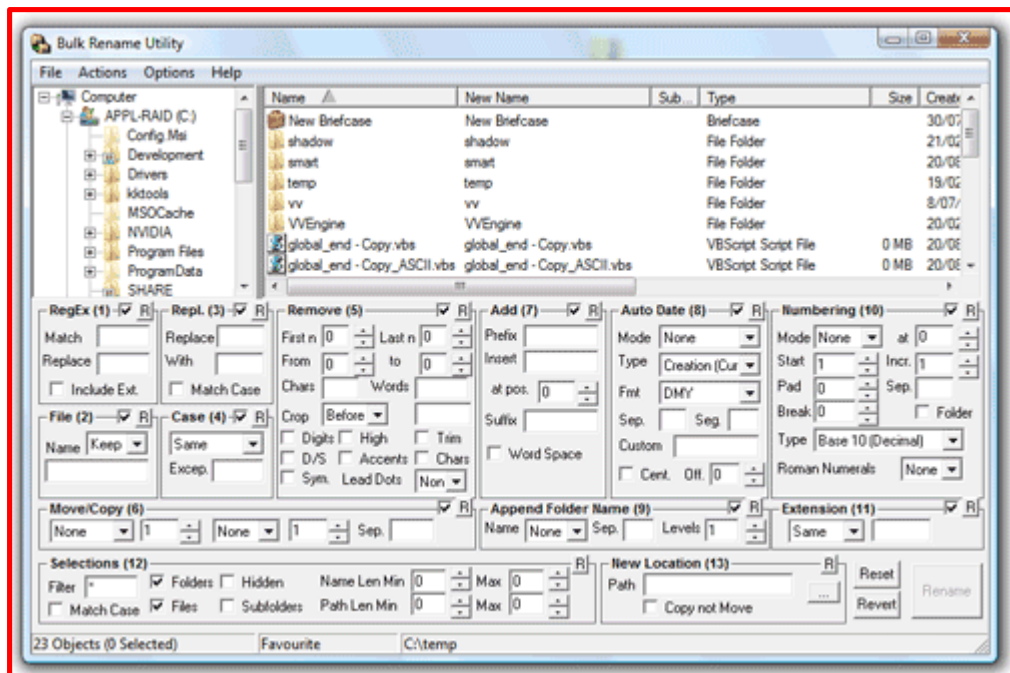


En las siguientes imágenes tenemos dos interfaces de sendos programas para renombrar ficheros masivamente. *Bulk Rename Utility* tiene una interfaz claramente poco usable, dado que introduce todas las opciones en una única pantalla. En contraposición, *ReNamer* ofrece una interfaz con las opciones más utilizadas en la pantalla principal, y las avanzadas agrupadas en diferentes pantallas.

ASÍ SÍ:

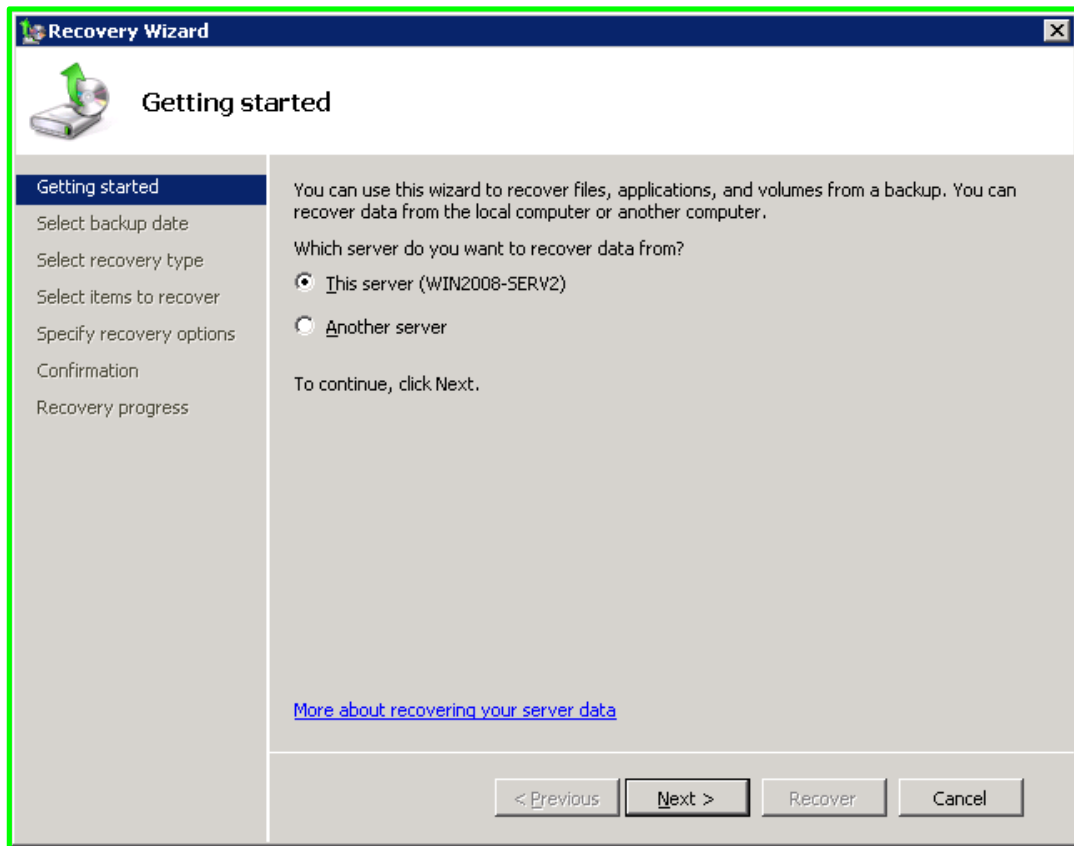


ASÍ NO:



6. **Diseñar diálogos para conducir a la finalización.** Crear sistemas claros de conducción hasta el final del proceso de las secuencias de acción del usuario. Un Wizard es un claro ejemplo de organización por pasos que delimitan una operación

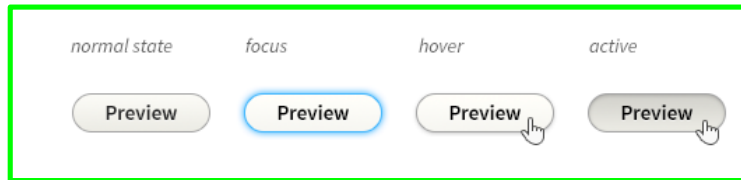
lineal (p.ej. una instalación). Cada vez que se proporciona una elección, se está pidiendo al usuario que tome una decisión. Esto significa que tendrán que pensar acerca de algo y decidir, lo cual no es necesariamente malo, pero en general, se debe intentar minimizar el número de decisiones que el usuario ha de tomar.



7. **Usar metáforas e invitaciones.** Las metáforas son similitudes con otros mecanismos y procesos conocidos por el usuario, generalmente de la vida real, que aplica lo que ya conoce a los elementos y relaciones dentro de un dominio no familiar (Autoexplicativa). El ejemplo más tradicional es el escritorio de Windows, que representa una “oficina virtual” con sus iconos, que son metáforas de carpetas y documentos físicos, o la papelera que representa la eliminación de ficheros (metáfora de tirar un documento a la basura).

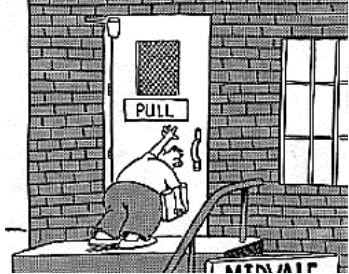


Las invitaciones son opciones visuales que nos “invitan” a realizar una acción sobre la interfaz. Por ejemplo, que el cursor cambie de forma al pasar sobre un enlace, o que un botón se “ilumine” (hover) son invitaciones.

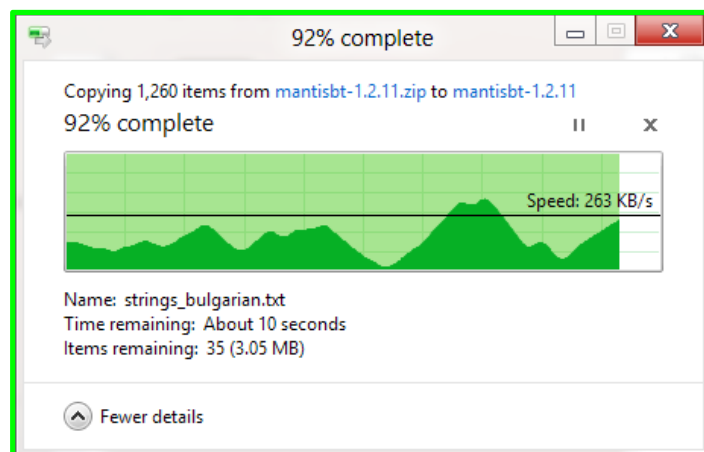
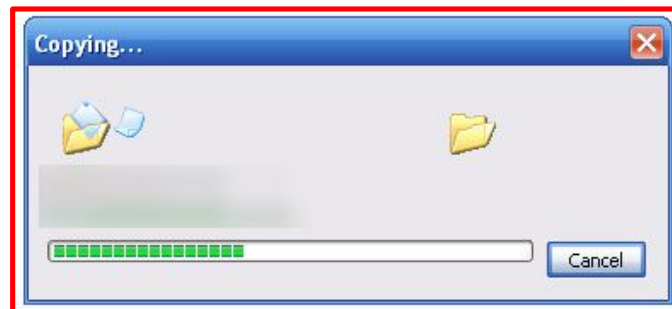


Puedes ver más sobre metáforas e invitaciones en [“Joel on Software La Opinión de Joel sobre qué es Software”](#).

Un buen ejemplo de interfaz sencilla que ha evolucionado últimamente para cumplir estos principios es la del diálogo de copiar ficheros en Windows. Con el paso del tiempo ha ido mejorando y:



1. Permite cancelar, pausar y reanudar la operación: da el control al usuario.
2. Informa del progreso: ahora no solo tiene una barra de progreso, sino que además muestra la tasa de transferencia para que el usuario pueda apreciar si se ha producido una bajada de velocidad en la transferencia y por eso está tardando más.
3. Tratamiento de errores consistente y homogéneo con el resto del sistema.



## Distribución de los elementos en pantalla

Una buena separación visual la podemos conseguir dejando espacios en blanco que hagan de bordes. Si queremos destacar algún elemento, podemos usar un borde con línea (véase “[Documentación oficial de Bordes en Swing](#)”). A la forma de distribuir los elementos en una pantalla la denominaremos “**Layouts**”.

JGoodies, empresa especializada en el diseño de interfaces con Swing, tiene [una guía](#) sobre los principios básicos de distribución de los elementos de un formulario. [Microsoft](#) nos dá sus propias guías para el diseño de formularios. [Lukew](#) tiene una guía mucho más extensa con ventajas e inconvenientes de cada interfaz, así como una selección de las mejores prácticas de diseño.

The screenshot shows a Java Swing window titled "Segment 1". The form is organized into three main sections: "Segment", "Diameters", and "Criteria". Each section contains several input fields. The layout uses equal-width columns to group related fields, such as "Identifier", "PTI [kW]", and "len [mm]" in the first column, and "Power [kW]" in the second column. The "Diameters" section has four columns for "da [mm]", "da2 [mm]", "R [mm]", "di [mm]", "di2 [mm]", and "D [mm]". The "Criteria" section has two columns for "Location", "Connection", "Holes", and "Slots". The "Holes" and "Slots" fields are checkboxes with labels "Has radial holes" and "Has longitudinal slots" respectively. The "Location" and "Connection" fields are dropdown menus with values "Propeller nut thread" and "C45E, ReH=600" respectively. The "k-factor" field is a text input field.

Section	Field	Value
Segment	Identifier	Segment 1
	PTI [kW]	
	len [mm]	
Diameters	da [mm]	
	da2 [mm]	
	R [mm]	
	di [mm]	
	di2 [mm]	
	D [mm]	
Criteria	Location	Propeller nut thread
	Connection	C45E, ReH=600
	Holes	<input checked="" type="checkbox"/> Has radial holes
	Slots	<input type="checkbox"/> Has longitudinal slots
	k-factor	

Equal width indicate similar function



# Validación

La validación en formularios nos sirve para cumplir el principio de diseño 4: minimizar la posibilidad de cometer errores.



Para validar un control en **Swing** hay que extender la clase `InputVerifier`. Todos los componentes de Swing colaboran entre ellos a la hora de pasarse el foco de unos a otros. Si un componente gana el foco, primero le pregunta al componente que lo pierde (y en concreto a su `InputVerifier`) si se puede hacer la transmisión del foco. De esta forma, por ejemplo, si escribimos algo en el `TextField` y vamos a pulsar un `Button`, el `Button` antes de ganar el foco preguntará al `InputVerifier` del `TextField` si se puede cambiar el foco del `TextField` al `Button`. Este es el momento que tiene el `TextField` para verificar si su entrada es o no correcta y negarse a perder el foco hasta que lo sea. El resultado es que podemos escribir cosas incorrectas, pero no podremos salir del `TextField` hasta que la entrada sea correcta.

El `InputVerifier` solamente tiene el método `verify`, que devuelve un boolean. Debe devolver "true" si la verificación es correcta - y por tanto podrá pasar a otro control - y "false" si la verificación es incorrecta.

En el siguiente ejemplo, se muestra una verificación de un DNI a través de un [Regex](#) de java. Si no se verifica que el DNI escrito es correcto, devuelve false.

```
class PassVerifier extends InputVerifier {
    Pattern dni = Pattern.compile("(\\d{8})([-]?)([A-Z]{1})");
    // Verifica que la contraseña sea la correcta
    public boolean verify(JComponent input) {
        JTextField tf = (JTextField) input;
        return dni.matcher(texto).matches();
    }
}
...
textField.setInputVerifier( new PassVerifier() );
```

Otra posible solución más respetuosa con el usuario (**principio 1, adoptar el punto de vista del usuario**) es la de devolver siempre "true" y mostrar un mensaje de error. De esta manera

permitimos que el usuario pueda salir del campo de texto sin introducir un valor válido (que puede que no sepa introducir), y le informamos del error.

```
class PassVerifier extends InputVerifier {
    Pattern dni = Pattern.compile("(\\d{8})([-]?)([A-Z]{1})");
    // Verifica que la contraseña sea la correcta
    public boolean verify(JComponent input) {
        JTextField tf = (JTextField) input;

        // Muestra error si no verifica
        labelError.setVisible( dni.matcher(texto).matches() );

        // Siempre devuelve true
        return true;
    }
}
...
textField.setInputVerifier(new PassVerifier());
```

Swing también dispone de APIs de validación de terceros. Un ejemplo de ello es el Simple Validation API: <https://kenai.com/projects/simplevalidation/pages/Home>

JavaFX no tiene validación integrada. Para poder usar validación y decoración (iconos de validación) en FX, debemos usar la librería externa [ControlsFX](#), que a través de su [API](#) de validación, nos proporciona la clase [ValidationSupport](#) que permite añadir un validador a un control a través del método [registerValidator](#).

También podemos diseñar en JavaFX nuestra propia validación con CSS, tal y como se explica en StackOverflow: <http://stackoverflow.com/questions/23579438/form-validator-message>

# Internacionalización

Otro aspecto importante de la usabilidad es tener en cuenta que nuestro programa puede ser usado por personas que no hablan nuestro idioma. A esto se le conoce como [internacionalización](#) y se suele abreviar con el acrónimo i18n (entre la primera i y la última n de internationalization hay 18 letras).

No es necesario que reescribamos el código de nuestra aplicación para traducirla: basta con evitar escribir directamente los Strings en el código (*hard-coded*). En su lugar, utilizaremos valores que se cogen de un fichero de propiedades. De esta manera:

1. Tendremos un fichero de propiedades para cada idioma
2. En base al idioma seleccionado, se mostrará la interfaz en ese idioma, cogiendo los valores del fichero asociado al mismo

Los ficheros de lenguaje son ficheros de propiedades de java. Esto es, ficheros de texto plano, que expresan el valor de las propiedades separados por líneas. Es decir, cada propiedad está en una nueva línea, y se expresa su valor con el símbolo igual (=). De esta manera, para acceder al texto de saludo, siempre accederemos a través del mismo string o propiedad (ej. greetings), y el fichero nos devolverá el valor en el idioma local.

MessagesBundle.properties	MessagesBundle_fr_FR.properties
<code>greetings = Hello</code> <code>farewell = Goodbye</code> <code>inquiry = How are you?</code>	<code>greetings = Bonjour.</code> <code>farewell = Au revoir.</code> <code>inquiry = Comment allez-vous?</code>

En el siguiente ejemplo podemos ver un sencillo código para saludar en varios idiomas. El programa toma dos argumentos (lenguaje y país), y muestra el saludo en el idioma escogido.

```
public void main(String args[]) {  
    // El programa toma dos argumentos: lenguaje y país  
    String language = args[0];  
    String country = args[1];  
    // Establezco el locale (idioma local)  
    Locale currentLocale = new Locale(language, country);  
}
```

```

// Obtengo el fichero con las frases para ese idioma
// El nombre base del fichero es "MessagesBundle"
ResourceBundle messages =
    ResourceBundle.getBundle("MessagesBundle", currentLocale);

System.out.println(messages.getString("greetings"));
System.out.println(messages.getString("inquiry"));
System.out.println(messages.getString("farewell"));
}

```

La salida para ese programa dependerá del idioma seleccionado:

% java I18NSample en US	% java I18NSample fr FR
Hello.	Bonjour.
How are you?	Comment allez-vous?
Goodbye.	Au revoir.

También es posible tener mensajes [compuestos por parámetros](#) y manejar [plurales](#). Asimismo, NetBeans [ayuda a internacionalizar](#) un formulario existente automáticamente.

Para mejorar la experiencia de uso de nuestra aplicación, sería recomendable guardar las opciones que elige el usuario, en lugar de preguntarle cada vez que inicia la aplicación el idioma en el que quiere trabajar. Para esta tarea, Java tiene una API que permite guardar las preferencias del usuario: [Java Preferences API](#). Esta librería realiza el “trabajo sucio” por nosotros y, en cada sistema, utiliza un fichero o el registro para guardar las preferencias del usuario sobre nuestra aplicación. En concreto, se guardan en el directorio del usuario:

- Linux: /home/user/.nombre\_aplicacion
- WXP: C:\Documents and Settings\user\
- W7: C:\Users\user\AppData\Local

La clase principal de este api es Preferences, y tiene tres métodos: put, get y remove, que nos se permite introducir un conjunto “clave-valor”, obtener un valor dada una clave y borrar la entrada respectivamente.

```
// Obtengo las preferencias del usuario
Preferences userPreferences = Preferences.userRoot();

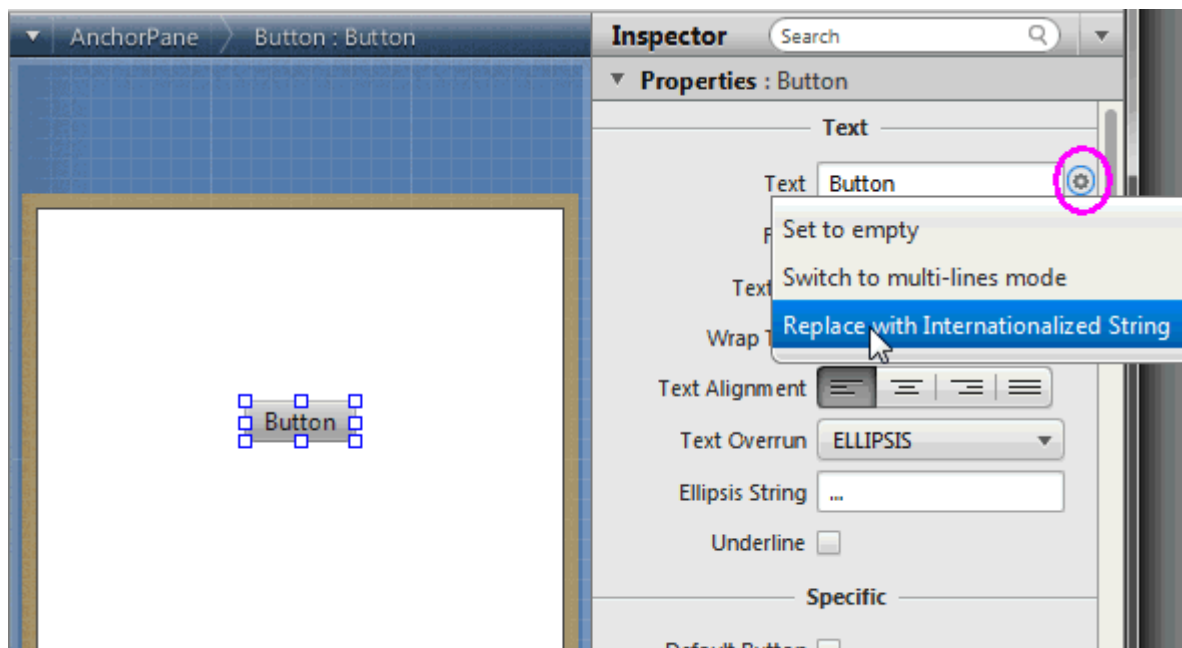
// Guardo un valor en preferencias
userPreferences.put("LANG", "ES");

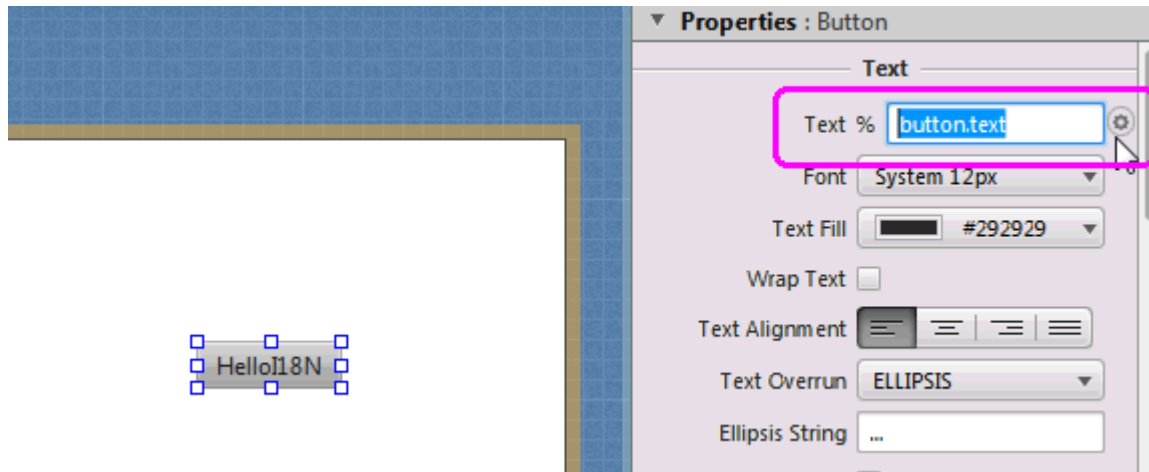
// Obtengo un valor de preferencias. Valor "ES" por defecto si no existe
String lang = userPreferences.get("LANG", "ES");

// Borro un valor de preferencias
userPreferences.remove("LANG");
```

## Internacionalización en JavaFX

En FX es posible internacionalizar estableciendo para cada control un texto internacionalizado (empiezan con el símbolo %):





```
<Button text="%button.text"/>
```

Podemos previsualizar el resultado seleccionando el fichero de idioma (ej. Bundle\_fr\_FR.properties) en el menú principal de **SceneBuilder Preview / Internationalization / Set Resource**.

En el controlador debemos establecer el bundle como parámetro del método initialize:

```
public class MyController implements Initializable {  
    @FXML private Label lblTextByController;  
    private ResourceBundle bundle;  
  
    @Override public void initialize(URL location, ResourceBundle resources) {  
        bundle = resources;  
    }  
}
```

Y para cargar la aplicación, debemos pasarle el bundle al FXMLLoader,

```
Locale locale = new Locale("en", "UK");  
ResourceBundle bundle = ResourceBundle.load("strings", locale);  
  
Parent root = FXMLLoader.load(  
    getClass().getClassLoader().getResource("ui/main.fxml"), bundle);
```

## Trabajo en segundo plano

Una de las necesidades para cumplir con **el principio de realimentación** es que la interfaz responda y dé información del progreso cuando está realizando tareas que son costosas en tiempo. Para ello, es necesario que creemos un hilo que atienda a estas tareas, pues el hilo principal (conocido como EDT o Event Dispatch Thread) es el que responde a las interacciones del usuario y no puede quedar bloqueado en una tarea larga.

Para esta finalidad Swing nos proporciona el [SwingWorker](#), una clase con las siguientes características:

- Define un método `doInBackground` que ejecuta el código en segundo plano.
  - Define otro método `get` que devuelve el resultado de la operación realizada. Este método tiene una implementación automática (no hay que redefinirlo) que devuelve lo mismo que se haya devuelto en el método `doInBackground`.
  - Define un método `done` (opcional) que se ejecuta cuando la tarea ha terminado.
  - Define un cuarto método (opcional) `process`, que se encarga de procesar los posibles resultados intermedios (*interims*) que obtenga el método.
- 
- Tiene dos parámetros para su construcción `SwingWorker<T1, T2>`.
    - El primero de ellos `T1` define el tipo de retorno del método `doInBackground` así como el método `get`
    - El segundo define el tipo de dato que toma como parámetro el método `process`

Veamos un ejemplo. Es muy típico, en las aplicaciones que trabajan con imágenes, que éstas se vayan cargando poco a poco, y que la interfaz responda desde el primer momento, en lugar de esperar a que se hayan cargado todas las imágenes para devolver el control al usuario. Vamos pues a definir nuestro Worker:

1. El trabajo consiste en cargar un conjunto de imágenes, por lo que el tipo de retorno del Worker (`T1`) será un array de imágenes (`ImageIcon[]`)
2. Cada vez que cargue una imagen se avisará a la interfaz de que la misma está disponible para ser mostrada, por lo que el tipo de dato “intermedio” (`T2`) será una imagen (`ImageIcon`)

```
SwingWorker worker = new SwingWorker<ImageIcon[], ImageIcon>() {  
    @Override
```

```

    public ImageIcon[] doInBackground() {
        final ImageIcon[] innerImgs = new ImageIcon[nimgs];
        for (int i = 0; i < nimgs; i++) {
            innerImgs[i] = loadImage(i+1);
            publish(innerImgs[i]);
            setProgress(100 * i / nimgs);
        }

        return innerImgs;
    }

    @Override
    public void done() {
        try {
            imgs = get();
        } catch (InterruptedException ignore) {}
    }

    protected void process(List<ImageIcon> images) {
        showImages(images);
    }
}

```

El método `process` recibe una lista de `ImageIcon` en lugar de una sola imagen. Esto se debe a que Swing llama al método no cada vez que se llama al método `publish`, sino cuando se han procesado un número determinado de resultados intermedios. Swing hace su propia planificación y va pasando resultados “cuando tiene tiempo”.

Es importante destacar que, dado el diseño de Swing, por cuestiones de sincronización de hilos, la interfaz sólo se debe modificar desde el EDT. Los métodos `process` y `done` se ejecutan en dicho hilo y, por tanto, se puede modificar la interfaz gráfica desde ellos. **NO se puede modificar la GUI desde el método `doInBackground`.**

Por último, para iniciar el worker solo debemos llamar al método `execute`:

```

SwingWorker worker = new SwingWorker(...);
worker.execute();

```



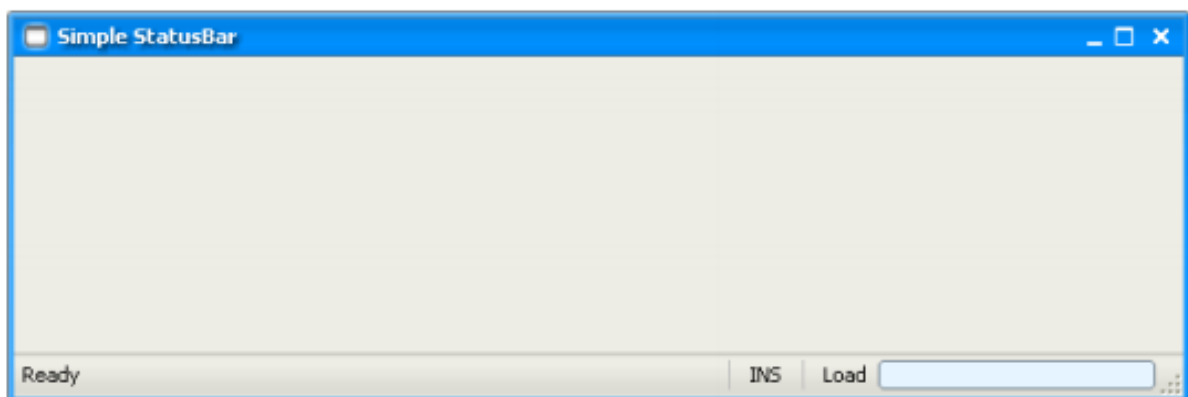
El worker también permite establecer / consultar el progreso de una tarea (métodos `setProgress` / `getProgress`), como un número que val de 1 al 100. De hecho en el ejemplo anterior se llama al método `setProgress`. `Progress` es una propiedad que lanza un evento cada vez que cambia (`setProgress`). La forma de conocer el progreso de una tarea sería suscribirnos a ese evento. Podemos crear una interfaz que se suscriba a dicho evento y modifique una [barra de progreso](#) cada vez que el progreso cambie:

```
progressBar = new JProgressBar(0, 100);
progressBar.setValue(0);
progressBar.setStringPainted(true);

worker.addPropertyChangeListener((evt) -> {
    if ("progress" == evt.getPropertyName()) {
        int progress = (Integer) evt.getNewValue();
        progressBar.setValue(progress);
    }
});
```

Es muy común que la barra de progreso esté integrada en una barra de estado o *status bar*. Aunque Swing no implementa este control directamente, podemos conseguirlo utilizando el `borderLayout` para añadir un pequeño panel en el que poner labels y progress bar.

```
getContentPane().add(statusBarPanel, java.awt.BorderLayout.SOUTH);
```



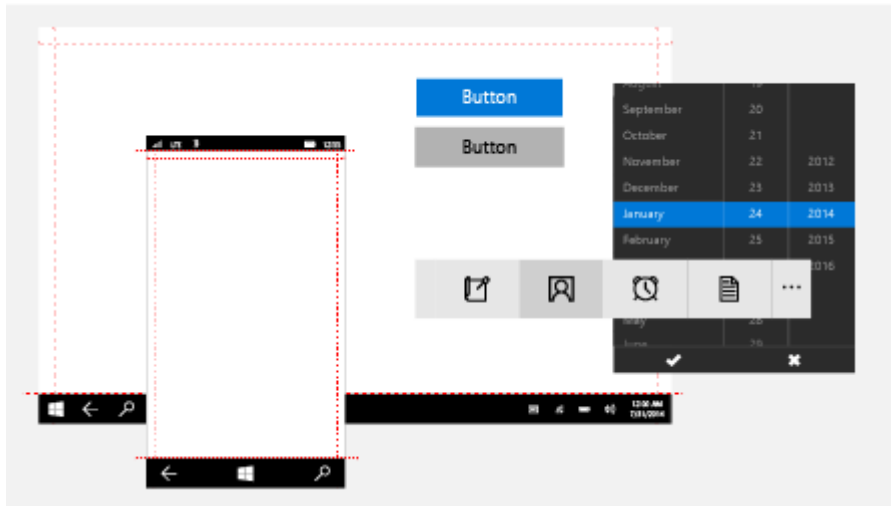
En JavaFX también existe la clase [Worker](#) con funcionamiento similar.

## Guías de diseño (Ribbon, Lolipop, etc.)

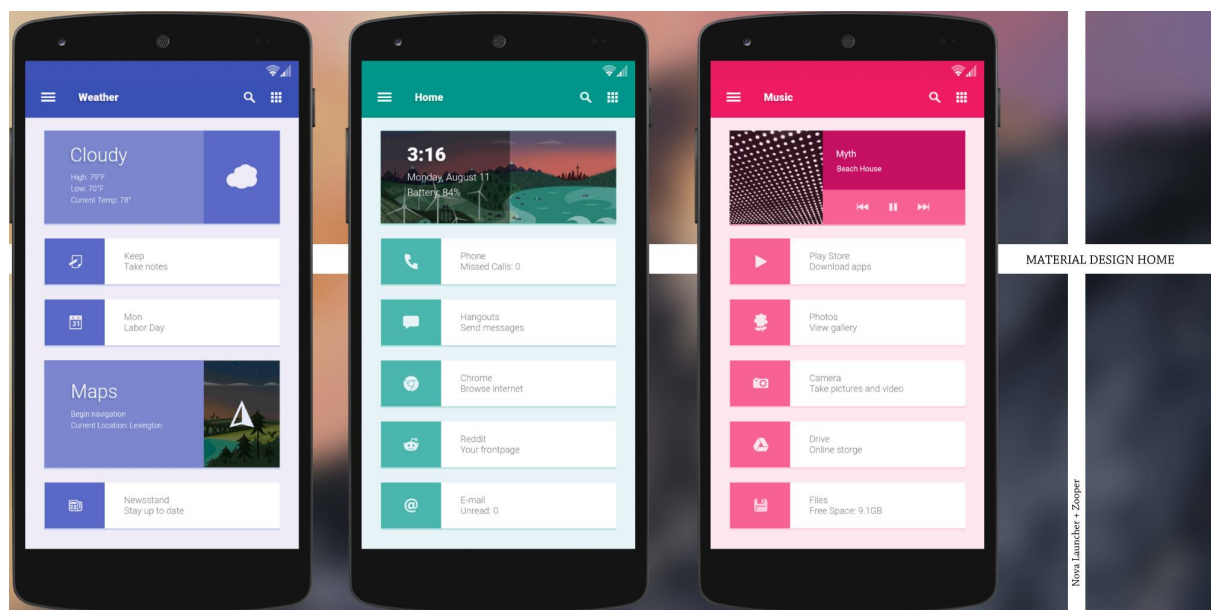
Es importante cuando estamos diseñando software para un sistema concreto intentar ceñirnos a las guías de diseño del mismo. Esto mejora sustancialmente la experiencia del

usuario, dado que respetamos el principio de consistencia y homogeneidad, no sólo en nuestra aplicación, sino a lo largo de todo el sistema operativo.

**Microsoft** fue pionera en esto, fijando algo tan trivial como el orden de la barra de menús (Archivo, Editar, Ver, etc.). Hoy en día van mucho más allá y proporcionan [una guía de diseño](#) de aplicaciones UWP (Universal Windows Platform), que son aplicaciones Windows diseñadas para ejecutarse en cualquier dispositivo con Windows (portátil, sobremesa, tablet o smartphone).



**Apple** también tiene su propia [guía](#) y, por su parte, Google proporciona el llamado [Material Design](#) utilizado desde la versión **Lollipop** de **Android**.



## Enlaces de Interés

- Patrones aplicables a problemas de diseño <http://patternry.com/patterns/>

- Libro “Designing Interfaces”. Web con patrones <http://designinginterfaces.com/patterns/>
- Sobre los colores complementarios. <http://slideplayer.es/slide/1815828/>
- Lista de libros sobre diseño, agrupados por categoría: <http://uxdesign.cc/ux-books/>
- Diseños para inspirarnos <http://zurb.com/patterntap>, <http://www.deeziner.com/> y <http://pttrns.com/android-patterns>

# Accesibilidad

## Concepto

*La accesibilidad es la cualidad de aquello que resulta accesible. El adjetivo accesible, por su parte, refiere a lo que es de comprensión o entendimiento sencillo.*

El concepto de accesibilidad, por lo tanto, se utiliza para nombrar al grado o nivel en el que cualquier ser humano, **más allá de su condición física o de sus facultades cognitivas**, puede usar una cosa, disfrutar de un servicio o hacer uso de una infraestructura.

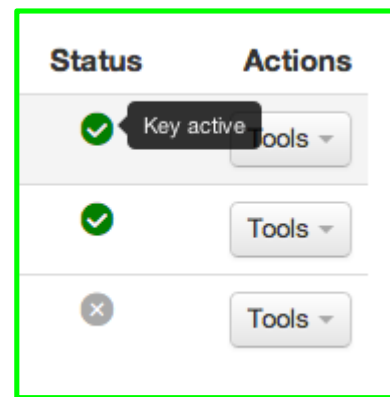
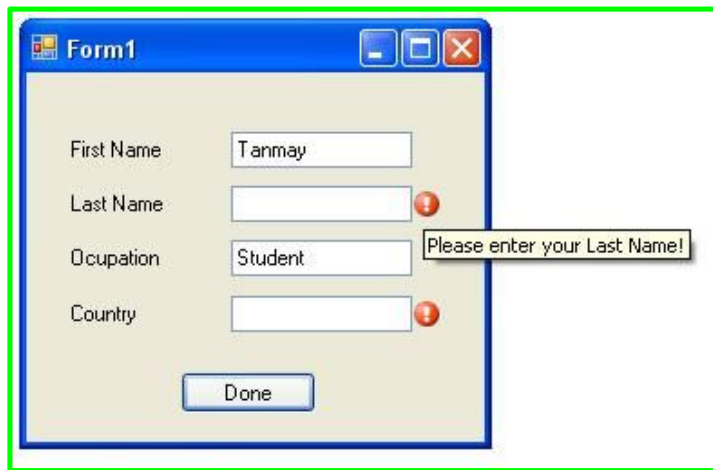
Existen distintos tipos de ayudas técnicas para discapacitados: teclados / ratones adaptados, lectores de pantalla o dispositivos de reconocimiento de voz. Este tipo de hardware, conocido como *tiflotecnología* depende de que el software sea accesible para poder funcionar adecuadamente.

El W3C tiene una buena [guía de accesibilidad web](#), extensible también a cualquier tipo de aplicación, en la que define los niveles de accesibilidad (desde nada accesible hasta totalmente accesible) y los puntos que hay que verificar en nuestra app.

## Accesibilidad

Como desarrolladores de software, tenemos que seguir unos principios a la hora de diseñar aplicaciones para hacerlas accesibles. La accesibilidad en Swing y en JavaFX viene integrada en el framework. Por ejemplo, una ayuda técnica puede obtener directamente el texto de una etiqueta o botón, sin necesidad de hacer nada especial. Sin embargo, es necesario seguir una serie de reglas para asegurar la accesibilidad de nuestra aplicación

1. Si el componente no muestra texto (ej. una imagen), añadirle un nombre con el método `setAccessibleName`. Por ejemplo un botón que sólo tenga un icono, un panel que agrupa otros, etc.



2. Establecer el tooltip siempre que tenga sentido hacerlo

```
aJComponent.setTooltipText( "Clicking this component causes XYZ to happen.");
```

3. Si no se proporciona tooltip, añadir una descripción al control con el método `setAccessibleDescription`.

```
aComponent.getAccessibleContext()
    .setAccessibleDescription( "Clicking this component causes XYZ to happen.");
```

4. Asegúrate de que tu programa se puede usar sin el ratón (únicamente a través de teclado).
5. Agrupa los componentes de un grupo en un panel. Por ejemplo, los radiobutton pertenecientes a un `buttonGroup`.
6. Siempre que una etiqueta etiquete otro componente, utiliza el método `setLabelFor`.

```
firstNameLabel.setLabelFor(firstNameTextField);
```

# Prototipado

Para que nuestra aplicación cumpla los requisitos de la aplicación que propone el usuario es muy recomendable la realización de un prototipo VISUAL. Este prototipo debe contener un esquema de las pantallas que el usuario final verá cuando tenga su aplicación terminada.

Normalmente, el usuario final especifica de forma totalmente “pésima” lo que quiere y espera de su aplicación, luego para que el desarrollador pueda tener un control del proceso de desarrollo del mismo debe conocer el alcance del proyecto con anterioridad.

Los usuarios finales no entienden de términos específicos ni algoritmos que haya que implementar, pero lo que sí entienden es de lo que VEN. El INTERFACE es clave para el usuario final. Dentro del desarrollo de un proyecto podríamos encajar la construcción de un prototipo en la primera fase de toma de requisitos.



Un **prototipo** o **mockup** en inglés es una maqueta o modelo de un diseño o dispositivo para que nos hagamos una idea de cómo será el producto final. El prototipo puede ser muy útil para probar una funcionalidad concreta, para ver el aspecto de distintos diseños e incluso para realizar tests de usabilidad (UX) sin invertir tanto tiempo, esfuerzo o dinero como supondría de tratarse del producto final.

Aunque su origen está en el mundo físico, la creación de páginas web o de aplicaciones móviles también implica el uso de prototipos, ya que nos permite jugar con la colocación de los menús o elementos y con el diseño en general sin malgastar tiempo picando código.

De la misma manera que la creación de aplicaciones ha evolucionado mucho con el tiempo, el mismo camino ha seguido el diseño y elaboración de prototipos o mockups, con soluciones de escritorio u online para usar directamente desde el navegador web.

Hace años aparecieron propuestas muy interesantes de herramientas online de mockup como **Moqups** o **Balsamiq**.

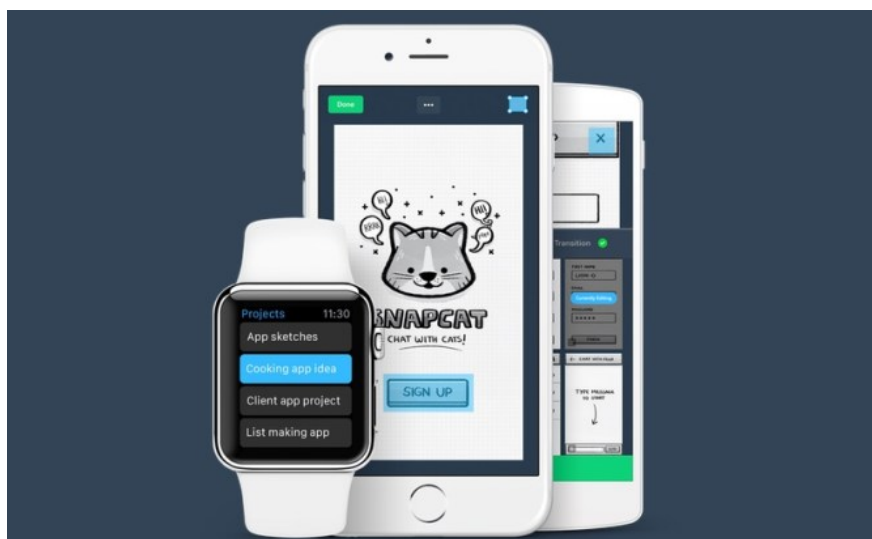
## Sitios Web para crear los mejores prototipos de páginas y apps móviles

### InVision

Centrada especialmente en el diseño, [InVision](#) sirve para crear mockups o prototipos tanto para páginas web como para aplicaciones iOS o Android.

Funciona desde cero o con nuestros diseños previos, que podemos subir a la herramienta. Con InVision podremos crear un prototipo con gestos, transiciones y animaciones, **prácticamente funcional** para probar cómo quedará finalmente.

Otra de sus ventajas es la posibilidad de **trabajar en solitario o en equipo** en el mismo diseño. Además, se lleva bien con otros servicios como Slack, Drive, Dropbox o Evernote para así **importar y exportar** información o parte del proyecto.



## Marvel

Con un nombre sugerente, [Marvel](#) se ofrece para que crees prototipos web o de apps móviles. Como ocurre con InVision, podrás complementar el uso de Marvel **con otras herramientas** como Slack, Google Drive o Asana.

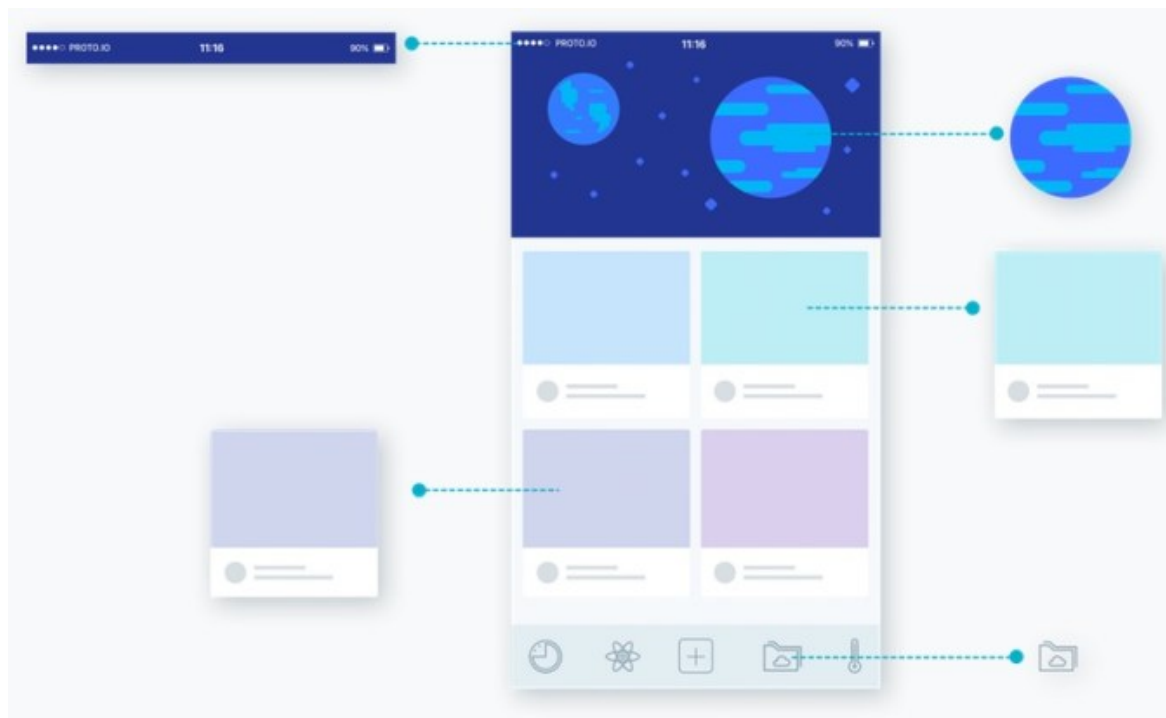
Además de trabajar en modo online, Marvel cuenta con [sus propias apps iOS y Android](#) para seguir diseñando desde tus dispositivos móviles.

Marvel te ayuda a crear el prototipo desde cero o a importar tus proyectos de Photoshop o Sketch. Luego podrás darle vida al mockup con **gestos y transiciones** de la librería de Marvel.

## Mockup Builder

Con **más de 800 plantillas** disponibles, [Mockup Builder](#) es un gran aliado si necesitas crear con poco tiempo un prototipo para páginas web, herramientas de escritorio o apps móviles.

En cualquier caso, en el editor encontrarás todo lo necesario para **crear menús, añadir botones y elementos interactivos** y diseñar todas las pantallas o páginas del proyecto.



## Proto.io

Prototipos que parecen de verdad. Así se presenta [Proto.io](#), una herramienta online de creación de mockups y prototipos especialmente **pensado para aplicaciones móviles**.



Con la herramienta de creación de Proto.io podemos crear prototipos con animaciones y gestos que dan cuenta de **cómo será la app** definitiva. Además, cuenta con **elementos prediseñados** para ayudarnos y que el resultado sea atractivo con poco esfuerzo.

## NinjaMock

Para terminar, [NinjaMock](#), que sirve tanto para diseño de páginas web como para aplicaciones Android, iOS o Windows Phone.

NinjaMock **trabaja básicamente con vectores**, de tu propiedad o importados de otras fuentes. Luego podrás exportar el resultado en PDF o compartirlo a partir de su enlace.

Por otro lado, admite comentarios, muy práctico si trabajas en equipo en el mismo prototipo.