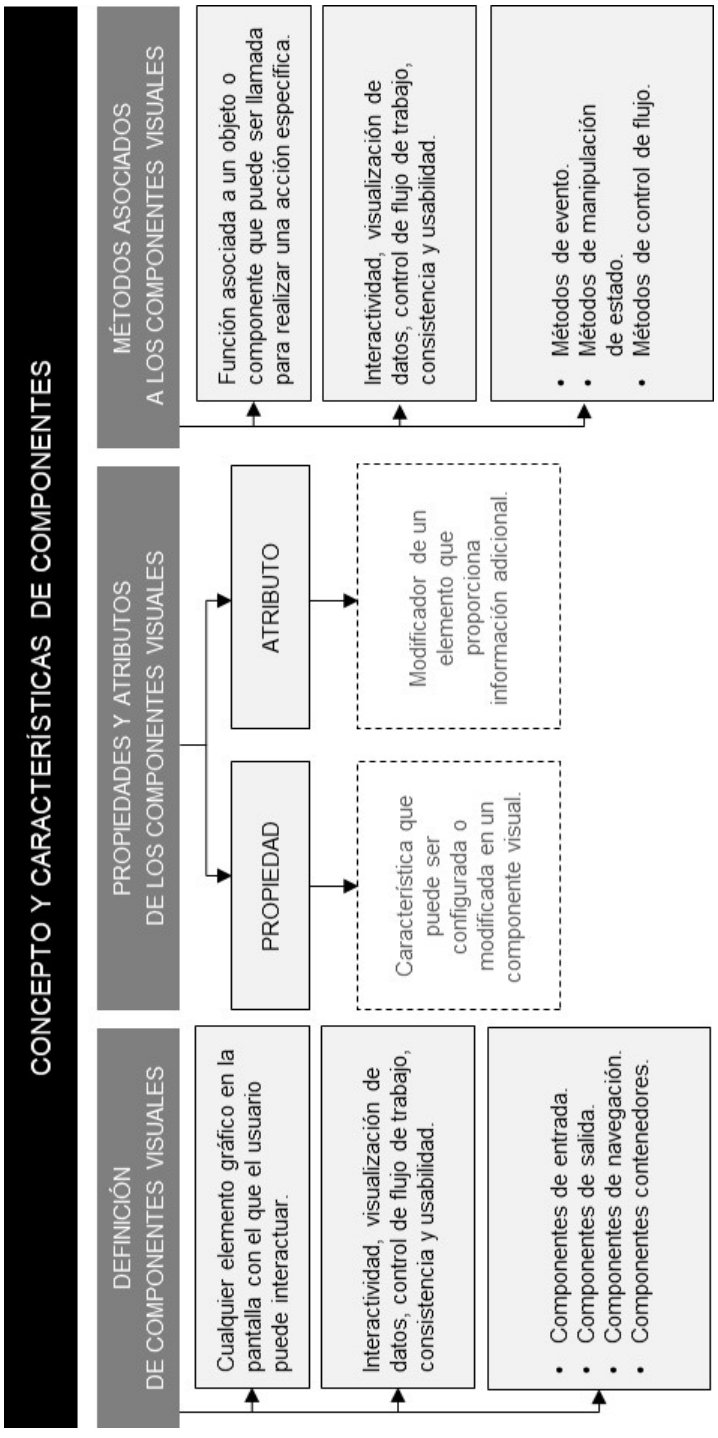
Desarrollo de Interfaces

Tema 1. Concepto y características de componentes

Índice


CONCEPTO Y CARACTERÍSTICAS DE COMPONENTES
DEFINICIÓN DE COMPONENTES VISUALES
Cualquier elemento gráfico en la pantalla con el que el usuario puede interactuar.
Interactividad, visualización de datos, control de flujo de trabajo, consistencia y usabilidad.
Componentes de entrada.
Componentes de salida.
Componentes de navegación.
Componentes contenedores.
PROPIEDADES Y ATRIBUTOS DE LOS COMPONENTES VISUALES
PROPIEDAD
Característica que puede ser configurada o modificada en un componente visual.
ATRIBUTO
Modificador de un elemento que proporciona información adicional.
MÉTODOS ASOCIADOS A LOS COMPONENTES VISUALES
Función asociada a un objeto o componente que puede ser llamada para realizar una acción específica.
Interactividad, visualización de datos, control de flujo de trabajo, consistencia y usabilidad.
Métodos de evento.
Métodos de manipulación de estado.
Métodos de control de flujo.

1.1. Introducción y objetivos

El desarrollo de **interfaces de usuario (UI)** es una disciplina fundamental en la creación de **aplicaciones** y **sistemas interactivos,** donde el objetivo principal es facilitar la **interacción** entre los usuarios y la tecnología. A medida que la tecnología avanza, la necesidad de interfaces intuitivas y eficientes se vuelve cada vez más crucial, ya que una buena UI no solo mejora la experiencia del usuario, sino que también influye en el éxito y la aceptación de un producto.

Los **componentes visuales** son los elementos básicos que conforman una interfaz de usuario. Estos componentes, como botones, cuadros de texto, menús y etiquetas, permiten a los usuarios interactuar con la aplicación de manera directa y comprensible. Cada uno de estos componentes tiene un conjunto de **propiedades** y **atributos** que definen su apariencia y comportamiento dentro de la interfaz y que permiten a los desarrolladores personalizar cómo se muestra y cómo responde cada componente a las interacciones del usuario. Comprender y manipular estas propiedades es esencial para crear interfaces que sean tanto funcionales como estéticamente agradables.

Además de las propiedades, los componentes visuales están asociados con **métodos específicos** que controlan su comportamiento en tiempo de ejecución. Estos métodos, permiten que los componentes respondan a eventos de usuario, como clics de ratón o cambios en los datos de entrada. Los métodos asociados son fundamentales para el desarrollo de aplicaciones dinámicas y adaptativas, donde la UI se adapta y reacciona de manera inteligente a las acciones del usuario.

Por otro lado, la **validación de entradas de usuario,** el **manejo de eventos** y la **manipulación de la interfaz en tiempo real** son aspectos esenciales que los desarrolladores deben dominar. Por ejemplo, la capacidad de validar un formulario antes de que se envíe o de mostrar mensajes de error si ciertos campos no están completos son prácticas comunes que mejoran, significativamente, la usabilidad de una aplicación.

En resumen, el desarrollo de interfaces de usuario es una combinación de arte y ciencia que requiere una comprensión profunda tanto de los componentes visuales como de los métodos que los controlan. Desde la manipulación de propiedades básicas hasta la implementación de métodos complejos que gestionan la interacción del usuario, cada aspecto de la UI contribuye a crear experiencias de usuario que son intuitivas, atractivas y funcionales. A medida que la tecnología continúa evolucionando, la importancia de interfaces bien diseñadas seguirá siendo un componente crucial en el desarrollo de *software.*

Los **objetivos** específicos que se pretenden alcanzar con este tema son:

- **Comprender el concepto de «componentes visuales»:** adquirir un conocimiento sólido sobre qué son los componentes visuales, su importancia en la creación de interfaces de usuario, y cómo se utilizan para construir aplicaciones interactivas y accesibles.

- **Manejo de propiedades y atributos de los componentes:** aprender a identificar y manipular las propiedades y atributos de los componentes visuales, como el texto, tamaño, color, la visibilidad y el estado habilitado/deshabilitado, para personalizar y mejorar la funcionalidad de la UI.

- **Aplicación de métodos asociados a los componentes:** desarrollar la habilidad de utilizar métodos específicos asociados a los componentes visuales para gestionar eventos, cambiar el estado de los componentes y controlar la interacción del usuario en tiempo real.

1.2. Historia y evolución de las interfaces de usuario (UI)

Las **interfaces de usuario (UI)** son un elemento fundamental en la **interacción** entre l a s **personas** y las **máquinas.** A lo largo de la historia, estas interfaces han evolucionado de manera significativa, adaptándose a las necesidades tecnológicas y a las expectativas de los usuarios. Desde las primitivas líneas de comando hasta las interfaces naturales y perceptuales actuales, el recorrido de las UI refleja la evolución tecnológica y la creciente importancia de la **usabilidad** y la **experiencia del usuario.** Este documento ofrece un recorrido detallado y actualizado sobre la historia y la evolución de las UI, desde sus inicios hasta la actualidad en 2024.

Las interfaces de usuario son los **medios** a través de los cuales los **usuarios interactúan** con los **sistemas informáticos.** Pueden definirse como la capa que conecta a las personas con los programas y dispositivos, permitiendo que los usuarios den órdenes y reciban respuestas. Las UI son esenciales en cualquier sistema que requiera interacción humana, ya que determinan cómo los usuarios perciben y manejan una aplicación o dispositivo.

Las interfaces se utilizan en una amplia variedad de contextos, desde sistemas operativos de **ordenadores** hasta **aplicaciones móviles,** y cada una está diseñada para **facilitar** la **comunicación** efectiva entre el usuario y el sistema.

Evolución de las interfaces de usuario

Interfaces de línea de comandos (CLI)

En los primeros días de la computación, la interacción entre el usuario y el sistema era puramente **textual.** Las interfaces de línea de comandos (CLI) eran la norma. Estas interfaces permitían a los usuarios **escribir comandos** en una consola para que **el sistema los ejecutara.** Un ejemplo emblemático es el sistema operativo MS-DOS, lanzado en 1982, que requería que los usuarios ingresaran comandos específicos para realizar tareas básicas.

Las CLI eran poderosas y ofrecían un control total sobre el sistema, pero su uso estaba limitado a usuarios con **conocimientos técnicos avanzados,** lo que las hacía inaccesibles para el público general.

Interfaces de usuario textuales (TUI)

Las interfaces textuales (TUI) representaron un avance respecto a las CLI al ofrecer una **estructura más organizada** para la interacción con el usuario. Estas interfaces seguían siendo basadas en texto, pero presentaban **menús** y **opciones** que **facilitaban** la **navegación.** Las TUI se utilizaban en configuraciones de BIOS y en sistemas operativos tempranos como UNIX, que presentaba una interfaz basada en texto para la gestión del sistema.

Aunque las TUI mejoraron la usabilidad en comparación con las CLI, seguían siendo **limitadas** en cuanto a la **facilidad de uso** y la **accesibilidad** para los usuarios menos técnicos.

Interfaces gráficas de usuario (GUI)

El verdadero cambio en las interfaces de usuario llegó con el desarrollo de las interfaces gráficas de usuario (GUI). Estas interfaces introdujeron **elementos visuales** como ventanas, iconos, menús y punteros, que facilitaban la **interacción** con el sistema de manera más **intuitiva** y **accesible.**

El primer sistema que implementó una GUI de manera efectiva fue el Xerox Alto en 1973, aunque no fue comercializado ampliamente. Apple popularizó la GUI con el lanzamiento del Macintosh en 1984, que presentó una interfaz amigable que incluía ventanas y el uso del *mouse* para interactuar con los elementos en pantalla. Microsoft siguió esta tendencia con el lanzamiento de Windows 1.0 en 1985, que evolucionó significativamente en versiones posteriores como Windows 3.0 y Windows 95, cada una mejorando la funcionalidad y la estética de las GUI.

Las GUI transformaron la computación al hacerla **accesible** a un **público más amplio,** eliminando la necesidad de aprender comandos complejos y permitiendo que los usuarios interactuaran con los sistemas a través de **representaciones visuales.**

Evolución en dispositivos móviles

Con el auge de los dispositivos móviles, las interfaces de usuario tuvieron que adaptarse a **pantallas más pequeñas** y a la **interacción táctil.** A principios de los 2000, los dispositivos como los primeros *smartphones* y PDA *(personal digital assistant)* comenzaron a incorporar interfaces gráficas simplificadas adaptadas a la interacción táctil.

El lanzamiento del **iPhone** en **2007** marcó un punto de inflexión en la historia de las UI móviles. Apple introdujo **iOS,** un sistema operativo con una GUI optimizada para pantallas táctiles, lo que facilitó la adopción masiva de *smartphones.* Paralelamente, **Google** lanzó **Android,** un sistema operativo de código abierto que también se centró en la interacción táctil, pero con una interfaz más personalizable. Ambos sistemas han evolucionado significativamente desde entonces, introduciendo conceptos como *material design* de Google y la estética minimalista de iOS.

Interfaces de voz y naturales (VUI y NUI)

A medida que la tecnología avanzaba, surgieron nuevas formas de interacción, como las interfaces de usuario de voz (VUI) y las interfaces naturales de usuario (NUI). Las **VUI** permiten a los usuarios **interactuar con dispositivos** a través de **comandos de voz,** una tecnología que se ha vuelto común con asistentes virtuales como Siri de Apple, Alexa de Amazon y Google Assistant.

Por otro lado, las **NUI** permiten la **interacción** mediante **gestos, movimientos** y **otras formas intuitivas de control.** Estas interfaces están diseñadas para ser lo más naturales posible, permitiendo que la tecnología responda de manera más directa a las acciones humanas. Ejemplos de NUI incluyen dispositivos como la consola Wii de Nintendo y sistemas de reconocimiento de gestos como Microsoft Kinect.

Interfaces perceptuales y cerebro-computadora (PUI y BCI)

Las interfaces perceptuales (PUI) y las interfaces cerebro-computadora (BCI) representan la vanguardia de la interacción humano-computadora. Las **PUI combinan** elementos de las **VUI** y **NUI** con tecnologías de **reconocimiento de gestos** y **emociones,** permitiendo una **interacción** más **inmersiva** y **personalizada.** Estas interfaces están en desarrollo y se utilizan en campos avanzados como la medicina y la rehabilitación.

Las **BCI,** por su parte, permiten la **comunicación directa** entre el **cerebro** y un **dispositivo externo,** utilizando ondas cerebrales para controlar interfaces. Aunque esta tecnología está en sus primeras etapas, ya ha mostrado promesas en aplicaciones para personas con discapacidades físicas, permitiendo, por ejemplo, el control de prótesis robóticas.

1.3. Definición de componentes visuales

Los **componentes visuales** son los elementos fundamentales que conforman la estructura de una interfaz de usuario (UI). Estos componentes, también conocidos como *widgets* o controles, son las piezas que permiten a los usuarios **interactuar** con una aplicación de manera intuitiva y efectiva. Los componentes visuales incluyen una amplia gama de elementos, como botones, cuadros de texto, menús desplegables, listas, barras de desplazamiento y muchos otros. Cada uno de estos componentes cumple una **función específica** dentro de la interfaz, facilitando la ejecución de tareas y la navegación dentro de la aplicación.

¿Qué son los componentes visuales?

En términos simples, un componente visual es cualquier **elemento gráfico en la pantalla** con el que el usuario puede **interactuar.** Estos componentes son la interfaz tangible que **conecta** al **usuario** con la **funcionalidad subyacente** de una aplicación. Por ejemplo, cuando usas un botón en una aplicación móvil para enviar un mensaje, ese botón es un componente visual que ejecuta una acción específica (enviar el mensaje) cuando se hace clic en él. De manera similar, un cuadro de texto permite a los usuarios ingresar datos que el sistema luego procesa.

Características de los componentes visuales

- **Interactividad:** los componentes visuales permiten la interacción directa entre el usuario y la aplicación. Esta interacción puede ser tan simple como hacer clic en un botón o tan compleja como arrastrar y soltar elementos dentro de una ventana.

- **Visualización de datos:** algunos componentes están diseñados para mostrar información al usuario, como etiquetas *(labels)* o cuadros de texto que muestran datos.

- **Control de flujos de trabajo:** otros componentes controlan la navegación dentro de la aplicación, como las pestañas o los menús que permiten moverse entre diferentes secciones o funciones de la aplicación.

- **Consistencia y usabilidad:** los componentes visuales suelen seguir patrones de diseño estándar que los usuarios ya conocen, lo que facilita la usabilidad y reduce la curva de aprendizaje.

Tipos de componentes visuales

Los componentes visuales pueden clasificarse en varias categorías según su función y la manera en que interactúan con el usuario:

- **Componentes de entrada:** permiten al usuario ingresar datos o comandos. Por ejemplo:
  - **Cuadro de texto:** permite la entrada de texto por parte del usuario.
  - **Botón:** inicia una acción específica cuando se presiona.
  - ***Checkbox:*** permite seleccionar o deseleccionar opciones.
  - ***Radio button:*** similar al *checkbox,* pero permite seleccionar solo una opción dentro de un grupo.
- **Componentes de salida:** muestran información al usuario sin requerir interacción.
  - ***Label*** **(etiqueta):** muestra texto estático que no se puede modificar.
  - ***Progress bar*** **(barra de progreso):** indica, visualmente, el progreso de una tarea.

- **Componentes de navegación:** facilitan la navegación dentro de la aplicación.
  - **Menús:** listas desplegables de opciones que el usuario puede seleccionar.
  - ***Tabs* (pestañas):** Permiten cambiar entre diferentes vistas o secciones dentro de la misma ventana.
- **Componentes contenedores:** organizan y agrupan otros componentes visuales.
  - **Paneles:** agrupan componentes relacionados, permitiendo la organización visual y funcional de la UI.
  - **Ventanas modales:** cuadros de diálogo que requieren la atención del usuario antes de continuar con otras actividades.

Propiedades y atributos de los componentes visuales

Cada componente visual tiene un conjunto de propiedades y atributos que definen su apariencia, comportamiento y función dentro de la UI. Estas propiedades pueden incluir:

- ***Text:*** define el texto que aparece en un componente como un botón o una etiqueta.
- ***Size* (tamaño):** establece las dimensiones del componente en la interfaz.
- ***Color:*** permite configurar los colores del fondo, del texto y de los bordes del componente.
- ***Enabled/disabled:*** determina si un componente está activo y puede ser interactuado o si está deshabilitado y es solo visible.
- ***Visible:*** indica si el componente está visible en la pantalla o si está oculto.

Estas propiedades pueden ser **modificadas** durante el desarrollo de la aplicación en el entorno de diseño o, dinámicamente, en tiempo de ejecución mediante código.

Vamos a ver cómo se combinan varios componentes visuales para crear un formulario de registro simple utilizando HTML *(hypertext markup language)* y JavaScript.

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Formulario de Registro</title>
</head>
<body>
    <h2>Formulario de Registro</h2>
    <form id="registroForm">
        <label for="nombre">Nombre:</label>
        <input type="text" id="nombre" name="nombre"><br><br>

        <label for="email">Correo Electrónico:</label>
        <input type="email" id="email" name="email"><br><br>

        <label for="password">Contraseña:</label>
        <input type="password" id="password" name="password"><br><br>

        <label for="terminos">Acepto los términos y condiciones</label>
        <input type="checkbox" id="terminos" name="terminos"><br><br>

        <button type="button" onclick="registrar()">Registrarse</button>
    </form>
</body>
</html>
```

Figura 1. Formulario de registro. Código HTML. Fuente: elaboración propia.

```javascript
function registrar() {
    var nombre = document.getElementById('nombre').value;
    var email = document.getElementById('email').value;
    var password = document.getElementById('password').value;
    var terminos = document.getElementById('terminos').checked;

    if (nombre && email && password && terminos) {
        alert("Registro completado con éxito!");
    } else {
        alert("Por favor, complete todos los campos y acepte los términos.");
    }
}
```

Figura 2. Formulario de registro. Código Javascript. Fuente: elaboración propia.

1.4. Propiedades y atributos de los componentes visuales

En el desarrollo de interfaces de usuario (UI), las propiedades y atributos de los componentes visuales juegan un papel fundamental. Estas **propiedades** y **atributos** son las características que definen cómo un componente **se ve, se comporta** y **responde** a las interacciones del usuario. Comprender cómo funcionan es esencial para los desarrolladores que desean crear interfaces eficientes, accesibles y fáciles de usar.

¿Qué son las propiedades y atributos?

Las propiedades y atributos de un componente visual son **características configurables** que determinan su **apariencia** y **comportamiento** dentro de la interfaz de usuario. Las propiedades, generalmente, se configuran en el tiempo de diseño (cuando el desarrollador está construyendo la UI), pero también pueden modificarse dinámicamente en el tiempo de ejecución (cuando la aplicación ya está en uso).

Propiedad

Es una **característica** que puede ser **configurada** o **modificada** en un **componente visual.** Algunos ejemplos comunes incluyen el tamaño, el color, la visibilidad y el estado habilitado o deshabilitado de un componente.

Atributo

En el contexto de HTML y otros lenguajes de marcado, un atributo es un **modificador** de un elemento que proporciona información adicional. En muchos casos, los términos *propiedad* y *atributo* se utilizan de manera intercambiable, aunque los atributos suelen ser más estáticos y están presentes en el código fuente de la UI.

Principales propiedades de los componentes

- ***Text* (texto):** define el contenido de texto que aparece dentro de un componente. Por ejemplo, en un botón, la propiedad «Text» podría ser «Enviar» o «Aceptar».
- ***Size* (tamaño):** establece las dimensiones (ancho y alto) de un componente visual. Esto es crucial para adaptar el diseño a diferentes resoluciones de pantalla.
- ***Color:*** controla los colores de fondo, borde y texto de un componente. Por ejemplo, un botón puede tener un color de fondo azul y un texto blanco.
- ***Enabled* (habilitado/deshabilitado):** esta propiedad determina si un componente está activo y puede ser interactuado. Un botón deshabilitado, por ejemplo, no responde a los clics del usuario.
- ***Visible* (visibilidad):** define si un componente es visible en la interfaz o está oculto. Esto permite mostrar u ocultar componentes según la lógica de la aplicación.
- ***Position* (posición):** establece la ubicación del componente dentro de la interfaz. Los desarrolladores pueden posicionar un componente en una coordenada específica dentro de un contenedor o ventana.
- ***Alignment* (alineación):** controla cómo se alinea un componente dentro de su contenedor. Puede ser alineado a la izquierda, derecha, centro o justificado.

Consideremos un formulario simple en HTML que incluye un cuadro de texto, un botón y una etiqueta. Vamos a explorar cómo se pueden usar y manipular algunas propiedades básicas.

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Formulario de Ejemplo</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div id="miFormulario">
        <label for="nombre">Nombre:</label>
        <input type="text" id="nombre" name="nombre"><br><br>

        <button id="miBoton" onclick="validarFormulario()">Enviar</button>
        <p id="miLabel">Por favor, introduce un nombre.</p>
    </div>

    <script>
        function validarFormulario() {
            var nombre = document.getElementById('nombre').value;
            var label = document.getElementById('miLabel');
            if (nombre === "") {
                label.style.display = "block"; // Muestra el mensaje de error
            } else {
                label.style.display = "none"; // Oculta el mensaje de error
                alert("Formulario enviado correctamente!");
            }
        }
    </script>
</body>
```

Figura 3. Formulario simple. Código HTML y JavaScript. Fuente: elaboración propia.

```css
#miFormulario {
    width: 300px;
    margin: 50px auto;
    padding: 20px;
    border: 1px solid #ccc;
    background-color: #f9f9f9;
}

#miBoton {
    background-color: #4CAF50; /* Propiedad de color */
    color: white;
    padding: 10px 20px;
    border: none;
    cursor: pointer;
}

#miBoton:disabled {
    background-color: #ccc; /* Cambia el color si está deshabilitado */
    cursor: not-allowed;
}

#miLabel {
    display: none; /* Inicia oculto */
    color: red;
}
```

Figura 4. Formulario simple. Código CSS *(cascading style sheets).* Fuente: elaboración propia.

En este ejemplo, se configuran y manipulan varias propiedades de los componentes visuales para crear un formulario interactivo. El botón «Enviar» cambia de color si está deshabilitado, y la etiqueta de advertencia se muestra solo cuando el campo de texto está vacío.

Propiedades en otros entornos de desarrollo

Mientras que en HTML y JavaScript se usan atributos y propiedades CSS para controlar los componentes, en otros lenguajes de programación como C# o Java, las propiedades se gestionan directamente a través de la **sintaxis del lenguaje.**

```
Button miBoton = new Button();
miBoton.Text = "Enviar"; // Propiedad Text
miBoton.Width = 100; // Propiedad Width
miBoton.BackColor = System.Drawing.Color.Green; // Propiedad BackColor
miBoton.Enabled = false; // Propiedad Enabled (deshabilitado)
```

Figura 5. Formulario simple. Manejo de propiedades en C#. Fuente: elaboración propia.

En este código, las propiedades *text, width, backcolor* y *enabled* del botón son directamente manipuladas para definir cómo se presentará y funcionará el botón en la aplicación.

1.5. Métodos asociados a los componentes visuales

L o s **métodos asociados** a los componentes visuales son **funciones** o **procedimientos** que **permiten interactuar** con dichos componentes de manera dinámica. Estos métodos **definen** el **comportamiento** de los **componentes** y **cómo responden** a las **interacciones del usuario,** como clics de ratón, entradas de teclado o cambios en el estado de un componente. Es crucial comprender cómo estos métodos funcionan y cómo se utilizan para construir aplicaciones interactivas y adaptativas.

¿Qué son los métodos?

En el contexto de la programación y el desarrollo de interfaces de usuario, un método es una **función asociada a un objeto** o componente que puede ser llamada para realizar una **acción específica.** Los métodos permiten a los desarrolladores manipular los componentes en el tiempo de ejecución, respondiendo a eventos de usuario y modificando el estado o la apariencia del componente. Por ejemplo, un botón en una interfaz de usuario puede tener un método asociado llamado *onClick,* que define qué acción se ejecutará cuando el usuario haga clic en el botón.

Tipos comunes de métodos asociados a los componentes

- **Métodos de evento:** estos métodos se ejecutan en respuesta a una acción del usuario, como un clic de ratón, la presión de una tecla o el desplazamiento del ratón sobre un componente.

  - onClick() ***:*** método que se activa cuando se hace clic en un componente, como un botón.

  - onChange() ***:*** se activa cuando cambia el valor de un componente, como un cuadro de texto.

- onMouseOver() ***:*** se ejecuta cuando el cursor del ratón pasa por encima de un componente.

- **Métodos de manipulación del estado:** estos métodos permiten cambiar el estado de un componente, como su visibilidad, su contenido o si está habilitado o deshabilitado.

- show() **y** hide() **:** métodos que hacen visible o invisible un componente.

- enable() **y** disable() **:** métodos que habilitan o deshabilitan la interacción con un componente.

- setText() **:** establece el texto que se muestra en un componente, como un botón o una etiqueta.

- **Métodos de control de flujo:** estos métodos ayudan a gestionar la navegación y la estructura lógica de la interfaz.

- focus() **:** da el foco a un componente, permitiendo que el usuario interactúe directamente con él.

- blur() **:** quita el foco de un componente, desactivando su interacción directa.

El siguiente ejemplo muestra cómo los métodos asociados a los componentes permiten crear una interfaz dinámica e interactiva que responde en tiempo real a las acciones del usuario:

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Ejemplo de Métodos en Componentes</title>
    <script src="script_metodos.js"></script>
</head>
<body>
    <h2>Formulario Interactivo</h2>
    <div id="formulario">
        <label for="nombre">Nombre:</label>
        <input type="text" id="nombre" name="nombre" onfocus="cambiarFondo()" onblur="restaurarFondo()"><br><br>

        <button id="miBoton" onclick="mostrarNombre()">Mostrar Nombre</button>
        <p id="resultado"></p>
    </div>
</body>
</html>
```

Figura 6. Formulario simple. Código HMTL. Fuente: elaboración propia.

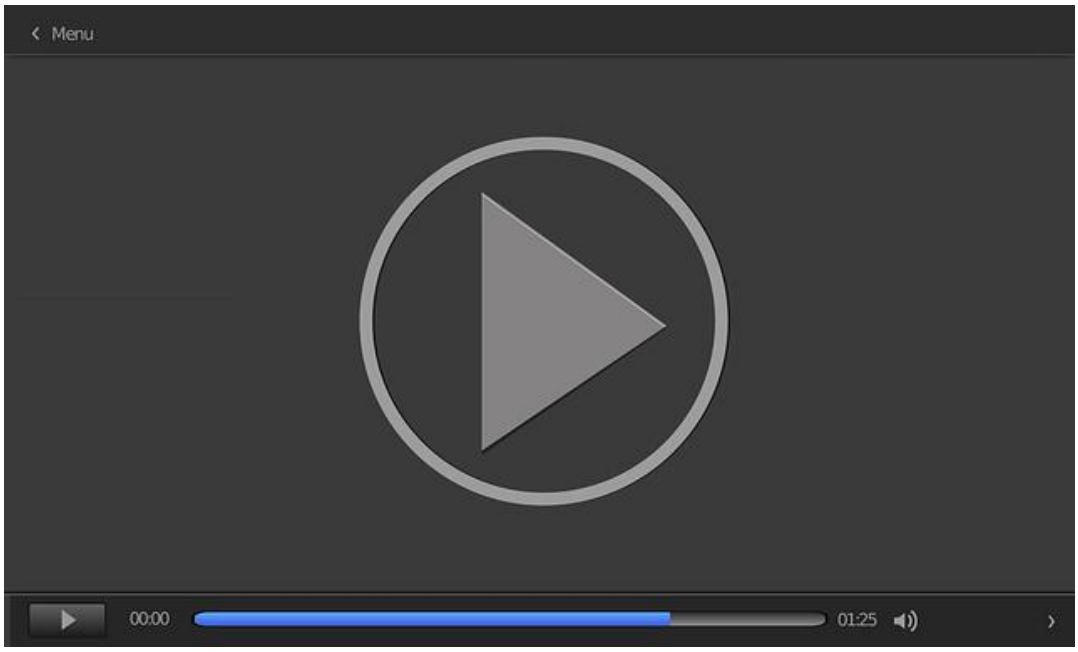
```
function cambiarFondo() {
    var campoNombre = document.getElementById('nombre');
    campoNombre.style.backgroundColor = "lightyellow";
}

function restaurarFondo() {
    var campoNombre = document.getElementById('nombre');
    campoNombre.style.backgroundColor = "";
}

function mostrarNombre() {
    var nombre = document.getElementById('nombre').value;
    var resultado = document.getElementById('resultado');

    if (nombre === "") {
        resultado.innerText = "Por favor, introduce tu nombre.";
        resultado.style.color = "red";
    } else {
        resultado.innerText = "Hola, " + nombre + "!";
        resultado.style.color = "green";
    }
}
```

Figura 7. Manejo de propiedades. Fuente: elaboración propia.

De la línea de comandos CLI a la interfaz gráfica de usuario GUI

Fredy Geek. (2020, agosto 27). *De la Linea de Comandos CLI a la Interfaz Grafica de Usuario GUI* [Vídeo]. YouTube. https://www.youtube.com/watch?v=bm-igX3rTwk&t=14s

Este vídeo explica todo lo relacionado con las interfaces gráficas de usuario, el entorno visual que nos permite interactuar con nuestro ordenador.



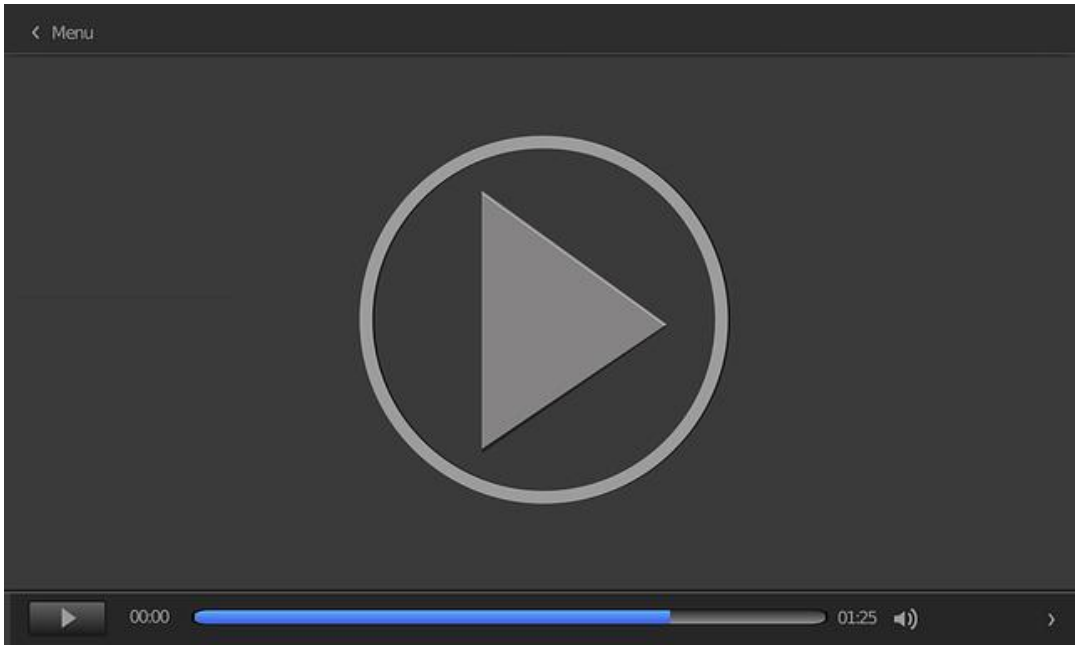
Accede al vídeo:

https://www.youtube.com/embed/bm-igX3rTwk

Componentes de las interfaces de usuario (GUI-Java)

Lewis García. (2021, marzo 23). *Componentes de las interfaces de usuario. GUI - JAVA* [Vídeo]. YouTube. https://www.youtube.com/watch?v=nQKupYnazqA

Este vídeo ofrece una introducción a los componentes de interfaces de usuario en Java.



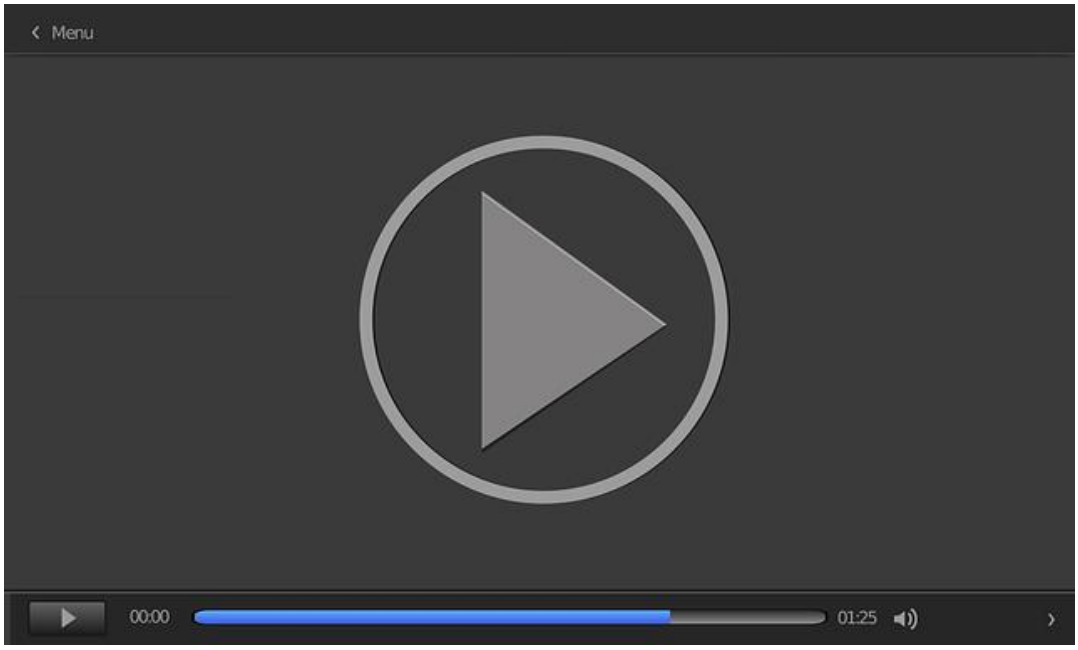
Accede al vídeo:

https://www.youtube.com/embed/nQKupYnazqA

Cómo diseñar una interfaz de login y registro en HTML y CSS

UGB. (2019, abril 8). *Como Diseñar una Interfaz de Login y Registro en HTML Y CSS* [Vídeo]. YouTube. https://www.youtube.com/watch?v=tNORFqbt-tc

En este vídeo se explica cómo diseñar una interfaz o formulario de *login* y registro usando HTML y CSS.



Accede al vídeo:

https://www.youtube.com/embed/tNORFqbt-tc

Entrenamiento 1

- Planteamiento del ejercicio

Crea una página web con un botón que cambie su color de fondo cuando el usuario pase el ratón sobre él y que vuelva a su color original cuando el ratón se aleje.

- Desarrollo paso a paso

  - Crea el archivo HTML: crea una estructura básica de HTML con un botón.
  - Agrega un evento `onMouseOver` : asocia un método que cambie el color del botón cuando el ratón pase sobre él.
  - Agrega un evento `onMouseOut` : asocia un método que restaure el color original cuando el ratón se aleje del botón.
  - Verifica el comportamiento: abre la página en un navegador y verifica que el botón cambie de color correctamente.

- Solución

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Botón Interactivo</title>
</head>
<body>
    <button id="miBoton" onmouseover="cambiarColor()" onmouseout="restaurarColor()">Pasa el ratón por aquí</button>

    <script>
        function cambiarColor() {
            document.getElementById('miBoton').style.backgroundColor = "lightblue";
        }

        function restaurarColor() {
            document.getElementById('miBoton').style.backgroundColor = "";
        }
    </script>
</body>
</html>
```

Figura 8. Solución. Fuente: elaboración propia.

Entrenamiento 2

- Planteamiento del ejercicio

Crea un formulario con un campo de texto para el nombre. Cuando el usuario intente enviar el formulario sin ingresar su nombre, debe mostrarse un mensaje de error en rojo.

- Desarrollo paso a paso

  - Crea el archivo HTML: crea un formulario con un campo de texto para el nombre y un botón de envío.
  - Agrega un evento `onClick` al botón: asocia un método que verifique si el campo de texto está vacío.
  - Muestra el mensaje de error: si el campo está vacío, muestra un mensaje de error en rojo.
  - Verifica la funcionalidad: prueba la validación ingresando y no ingresando un nombre.

- Solución

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Validar Formulario</title>
</head>
<body>
    <form id="miFormulario">
        <label for="nombre">Nombre:</label>
        <input type="text" id="nombre" name="nombre"><br><br>
        <button type="button" onclick="validarFormulario()">Enviar</button>
        <p id="mensajeError" style="color: red; display: none;">Por favor, introduce tu nombre.</p>
    </form>

    <script>
        function validarFormulario() {
            var nombre = document.getElementById('nombre').value;
            var mensajeError = document.getElementById('mensajeError');

            if (nombre === "") {
                mensajeError.style.display = "block";
            } else {
                mensajeError.style.display = "none";
                alert("Formulario enviado correctamente!");
            }
        }
    </script>
</body>
</html>
```

Figura 9. Solución. Fuente: elaboración propia.

Entrenamiento 3

- Planteamiento del ejercicio

Crea una página web con un botón que muestre u oculte un párrafo de texto al hacer clic en él.

- Desarrollo paso a paso

  - Crea el archivo HTML: escribe la estructura básica con un botón y un párrafo de texto.
  - Agrega un evento `onClick` al botón: asocia un método que oculte o muestre el párrafo según su estado actual.
  - Verifica el funcionamiento: prueba el comportamiento del botón al hacer clic en él.

- Solución

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Mostrar/Ocultar Contenido</title>
</head>
<body>
    <button onclick="toggleParrafo()">Mostrar/Ocultar</button>
    <p id="miParrafo" style="display: none;">Este es el párrafo de texto que se mostrará o se ocultará.</p>

    <script>
        function toggleParrafo() {
            var parrafo = document.getElementById('miParrafo');
            if (parrafo.style.display === "none") {
                parrafo.style.display = "block";
            } else {
                parrafo.style.display = "none";
            }
        }
    </script>
</body>
</html>
```

Figura 10. Solución. Fuente: elaboración propia.

Entrenamiento 4

- Planteamiento del ejercicio

Crea un botón que cambie su propio texto cada vez que el usuario hace clic en él.

- Desarrollo paso a paso

  - Crea el archivo HTML: crea un botón en la página
  - Agrega un evento `onClick` al botón: asocia un método que cambie el texto del botón cuando se haga clic en él.
  - Verifica el comportamiento: abre la página y verifica que el texto del botón cambie al hacer clic.

- Solución

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Cambiar Texto del Botón</title>
</head>
<body>
    <button id="miBoton" onclick="cambiarTexto()">Haz clic aquí</button>

    <script>
        function cambiarTexto() {
            var boton = document.getElementById('miBoton');
            boton.innerText = "Has hecho clic!";
        }
    </script>
</body>
</html>
```

Figura 11. Solución. Fuente: elaboración propia.

Entrenamiento 5

- Planteamiento del ejercicio

Crea un formulario con un campo de texto y un botón que esté deshabilitado hasta que el usuario ingrese un nombre en el campo de texto.

- Desarrollo paso a paso

  - Crea el archivo HTML: crea un formulario con un campo de texto y un botón.
  - Agrega un evento `onInput` al campo de texto: asocia un método que habilite el botón solo si el campo de texto no está vacío.
  - Deshabilita el botón por defecto: configura el botón para que esté deshabilitado cuando se carga la página.
  - Verifica la funcionalidad: prueba que el botón se habilite y deshabilite correctamente según si el campo de texto tiene contenido.

- Solución

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Formulario con Botón Deshabilitado</title>
</head>
<body>
    <form>
        <label for="nombre">Nombre:</label>
        <input type="text" id="nombre" name="nombre" oninput="verificarEntrada()"><br><br>
        <button id="miBoton" disabled>Enviar</button>
    </form>

    <script>
        function verificarEntrada() {
            var nombre = document.getElementById('nombre').value;
            var boton = document.getElementById('miBoton');

            if (nombre === "") {
                boton.disabled = true;
            } else {
                boton.disabled = false;
            }
        }
    </script>
</body>
</html>
```

Figura 12. Solución. Fuente: elaboración propia.