Algoritmos

Logro de sesión

Al finalizar la sesión, el estudiante hace uso de controles GUI del WinForms (con sus propiedades y eventos).

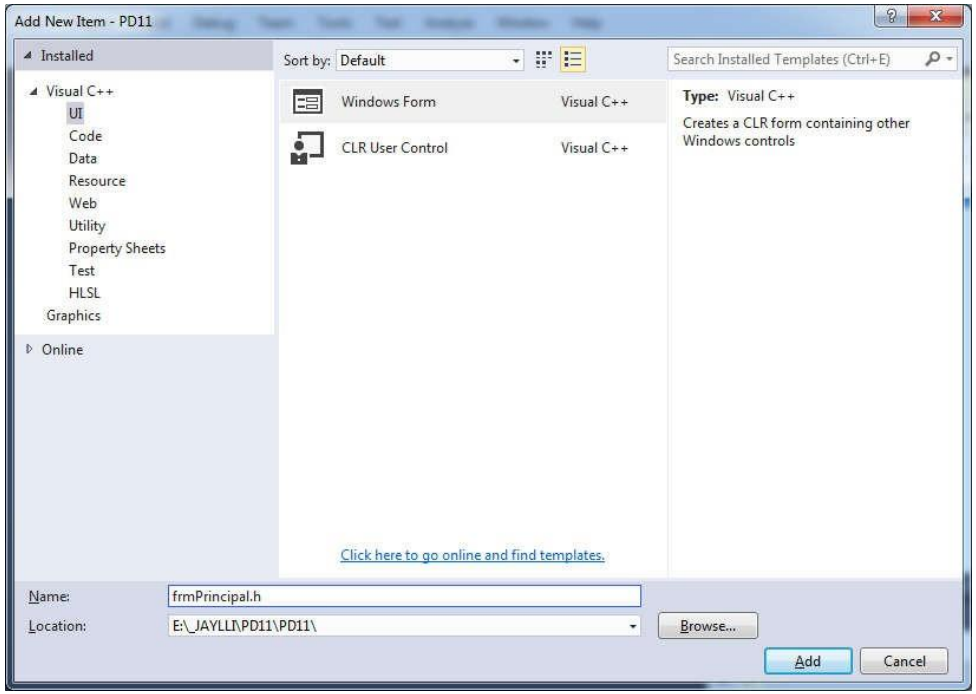
Formularios y Componentes Visuales

Contenido:

- Formularios
- Formularios MDI
- Componentes Visuales
- Creación Dinámica de Componentes
- Ejercicios

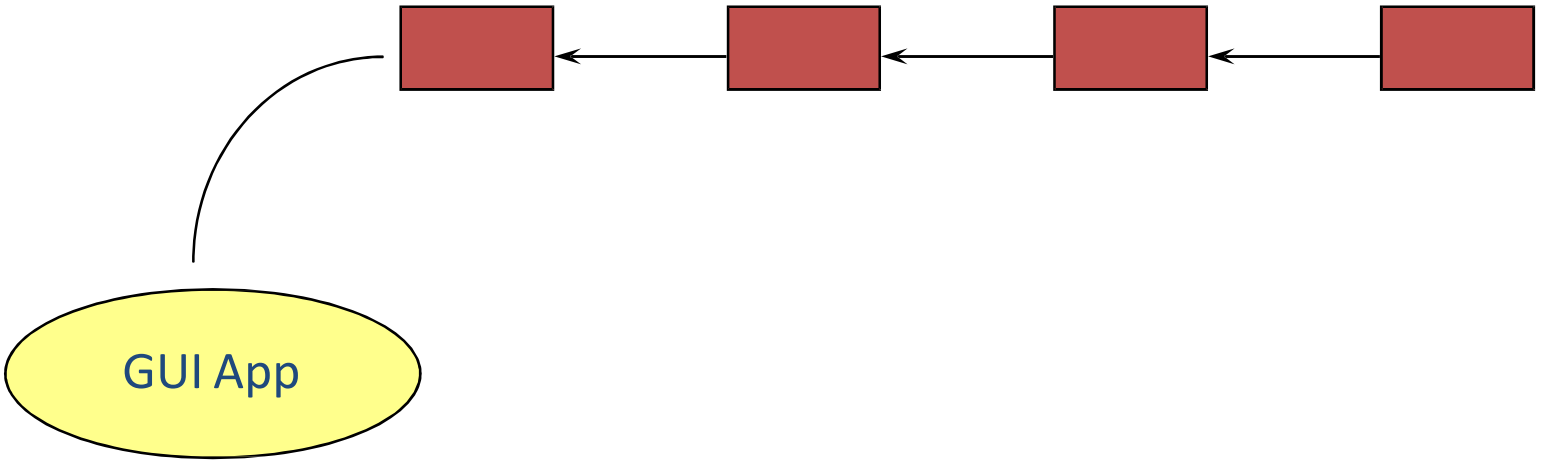
Aplicaciones Visuales

- *.NET Framework soporta dos tipos de aplicaciones basadas en formularios: WinForms y WebForms. Los WinForms son las aplicaciones GUI de escritorio tradicionales.*



Aplicaciones Basadas en Eventos

- Las acciones individuales de los usuarios se traducen en "eventos"
- Los eventos se pasan, 1 por 1, a la aplicación para el procesamiento.



Creación dinámica de componentes

- Los componentes son todos aquellos elementos que vemos en el Toolbox de Visual C++.
- Algunos de los componentes más utilizados son el botón, textbox, label etc.
- Hasta el momento, para poder agregar estos componentes a nuestro programa, simplemente los seleccionamos y los ponemos sobre el formulario.
- En esta sección definiremos como crearlos de forma dinámica mientras nuestro programa está en ejecución.

Los componentes son Clases

- Todos los **componentes** son **clases** como las que creamos en C++.
- Por ejemplo el **botón** que vemos en el formulario está representado por una clase llamada **Button** como se muestra a continuación.

```
ref class Button
{
   private:
      int Width;
      int Height;
      ...
   public:
      Button();
      void Click(System::Object^   sender, System::EventArgs^   e);
      ...
};
```

Creando un objeto de la clase Button

- Para crear un objeto de la clase **Button**, es necesario reconocerla como una **clase de .NET** por lo que debemos declarar un puntero al objeto utilizando ^.

```
Button ^miBoton = gcnew Button();
```

- Una vez creado este botón podemos cambiar todas sus propiedades para configurarlo como mejor nos parezca.

```
miBoton->Width = 150;
miBoton->Height = 25;
miBoton->Text = "Boton dinamico";
miBoton->BackColor = Color::Red;
```

Mostrando el componente en el formulario

- Sin embargo, el código anterior no es suficiente para que se muestre sobre el formulario.
- Hasta el momento el botón solo existe en memoria y no sabe dónde se debe mostrar.
- Para indicarle dónde se debe mostrar debemos decirle al botón cuál es su padre.
- **Una vez asignado el padre, el botón se mostrará en pantalla.**
- El padre de un botón puede ser cualquier componente que permita almacenar otro componente (Container) como por ejemplo un Formulario, un panel, un groupbox, etc.

Asignando el padre al componente

- Para asignarle el padre a un componente solamente es necesario colocar la propiedad Parent del mismo al valor de su padre.

```
// Para mostrarlo en el formulario
miBoton->Parent = this;

// Para mostrarlo en el panel 1
miBoton->Parent = panel1;
```

De que se encarga el Parent

- Al asignar la propiedad Parent se está asignando un **responsable** a nuestro componente.

- El Parent se encargará de dibujar, controlar y eliminar nuestro componente.

- Como el componente está dentro del Parent, es importante notar que toda su información es relativa al padre.
  - Un botón con coordenadas de posición (left, top) en (0,0) se encuentra en las coordenadas (0,0) de su padre.
  - Si fuese el formulario serían (0,0) pero de ser un panel serían (panel1->Left, panel1->Top)

Ejemplo para crear un botón aleatorio en pantalla.

- En algún evento del formulario, o inclusive en el evento clic de otro botón colocar el siguiente código:

```
Random ^r = gcnew Random();
Button ^miBoton = gcnew Button();
miBoton->Width = r->Next(20, 100);
miBoton->Height = r->Next(20, 100);
miBoton->Left = r->Next(this->ClientSize.Width);
miBoton->Top = r->Next(this->ClientSize.Height);
miBoton->Text = r->Next(10000).ToString();
miBoton->Parent = this;
```

Agregando eventos a componentes dinámicos

- Además de poder crear componentes de forma dinámica, es necesario poder configurar los eventos de estos componentes.

- Al configurar los eventos podemos hacer que el componente no solo aparezca en pantalla, sino que también responda ante los eventos como mejor nos parezca.

Agregando eventos a componentes dinámicos

- Para configurar un evento es necesario realizar 2 tareas:
  1. Crear un método con la misma estructura del evento a configurar.
  2. Asignar el método creado al evento correspondiente.
- Para conocer la estructura de los eventos podemos buscar en la ayuda de Visual C++ o simplemente copiarla de algún evento del código autogenerado.

Ejemplo del evento Click del botón

```
System::Void miClick(System::Object^ sender, System::EventArgs^ e)
{
  // Mostramos un mensaje y cambiamos el tamaño del botón presionado
  MessageBox::Show("Presionaste un boton dinamico");
  ((Button^)sender)->Width += 20;
}

System::Void Label1_Click(System::Object ^sender, System::EventArgs ^e)
{
    Random ^r = gcnew Random();
     // Creamos el botón de forma dinámica en este evento
     Button ^miBoton = gcnew Button();
     miBoton->Width = r->Next(20, 100);
     miBoton->Height = r->Next(20, 100);
     miBoton->Left = r->Next(this->ClientSize.Width);
     miBoton->Top = r->Next(this->ClientSize.Height);
     miBoton->Text = r->Next(10000).ToString();  miBoton-
     >Parent = this;
     // Le asignamos el evento que hemos creado llamado miClick
     miBoton->Click += gcnew System::EventHandler(this, &Form1::miClick);
}
```

Ejemplos con otros componentes

- Todos los componentes se pueden crear de forma dinámica utilizando el mismo procedimiento que el mostrado para el botón.

- Por ejemplo:

```
Label ^lbl = gcnew Label();
Panel ^pnl = gcnew Panel();
TextBox ^txt = gcnew TextBox();
```

- Inclusive podemos crear una serie de componentes dentro de un bucle:

```
for (int i=0; i < 10; i++)
{
    Button ^btn = gcnew Button();  btn->Left = i * 5;
    btn->Top = i * 5;  btn->Parent = this;
}
```

Formulario

- El formulario también es una clase llamada Form.
- Sin embargo, esta clase NO tiene una propiedad Parent que podamos cambiar (el parent siempre será Windows) debido a que el formulario es el contenedor de más alto nivel.
- Es por esta razón que las ventanas se muestran de forma independiente una de otra y no una dentro de la otra.
- En esta sección se verá como crear ventanas de forma dinámica.

Creando ventanas dinámicas

- Para crear una ventana dinámica, es necesario crearla como cualquier otro componente y en lugar de colocar el parent, llamamos al método Show.

```
Form ^miForm = gcnew Form();
miForm->BackColor = Color::Red;
miForm->Show();
```

Las ventanas siempre le pertenecen a nuestra aplicación de Windows que tiene como inicio al Form1.

Si el Form1 se cierra se cerrarán todas las ventanas creadas de forma dinámica.

Show vs. ShowDialog

- Show

- Muestra el formulario.
- Permite pasar a otra ventana.
- No interrumpe la ejecución de otro código.

USOS COMUNES:

- Ventanas emergentes (popup)
- Ver el detalle de un dato.
- Ventana de ayuda.

- ShowDialog

- Muestra el formulario.
- No permite pasar a otra ventana.
- Detiene el código siendo ejecutado hasta que el formulario se cierre.

USOS COMUNES:

- Cuadros de diálogo y confirmación.
- Configuración de opciones.

Ejemplo de Show y ShowDialog

- Show

- Coloque el siguiente código en el evento click de un bótón.

```
Form ^miForm = gcnew Form();
miForm->Show();
this->BackColor = Color::Red;
```

- El segundo formulario se muestra en pantalla y el color de nuestro formulario cambia a rojo.

- Coloque el siguiente código en el evento click de un bótón.

```
Form ^miForm = gcnew Form();
miForm->ShowDialog();
this->BackColor = Color::Red;
```

- El segundo formulario se muestra en pantalla y el color de nuestro formulario solo cambia cuando se cierra el segundo formulario.

Formularios personalizados

- No siempre es necesario crear todo de forma dinámica.
- Lo que se puede hacer es agregar un segundo formulario a nuestro proyecto.

  (Project -> Add new item -> Windows Form)
- Incluir la unidad de este formulario.
- Crear un objeto de dicha clase en lugar de crearla de tipo Form.

Creando un formulario personalizado.

1. Crear un Windows Form Application.
2. Seleccionar Project->Add new item->Windows Form.
3. Modificar el nuevo formulario (Form2) como mejor le parezca.
4. Ir al Form1.h e incluir el Form2: #include “Form2.h”
5. Ir al evento donde queremos crear el formulario y colocar:

```
Form2 ^frm = gcnew Form2();
frm->Show();
```

Pasando valores entre formularios

- Como el formulario es una clase, podemos implementar los métodos que queramos en ella dándonos así el control total sobre los mismos.

- Los atributos y métodos que implementemos tienen que seguir las mismas nórmas de encapsulamiento vistas para clases regulares, es decir, private y public dependiendo del tipo de dato.

Pasando valores entre formularios

- Tenemos 2 formularios (Form1 y Form2)
- En el Form1 hay 1 textbox, 1 label y un boton.
- En el Form2 hay un label y 2 botones.
- Cuando hagamos clic en el boton del Form1 queremos mandar el texto del textbox1 al Form2.
- El Form2 mostrara en el label un mensaje usando el texto recibido.
- El usuario presionará uno de los 2 botones del Form2.
- En el label del Form1 colocaremos el botón que presionó el usuario.

Código del Form1

- Agregar a la clase Form1 lo siguiente:

```
private:
 System::Void button1_Click(System::Object^ sender, System::EventArgs^ e) {
    // Creamos el formulario 2
    Form2 ^frmPreg = gcnew Form2();

    // Le pasamos el nombre ingresado
      frmPreg->ColocaNombre( textBox1->Text );

    // Mostramos el formulario 2 y esperamos hasta que se cierre
    frmPreg->ShowDialog();

    // Obtenemos la respuesta del formulario 2
     String ^s = frmPreg->DameRespuesta();

    // Colocamos la respuesta en el formulario 1  label1-
    >Text = "Respondiste " + s;
    delete frmPreg;
 }
```

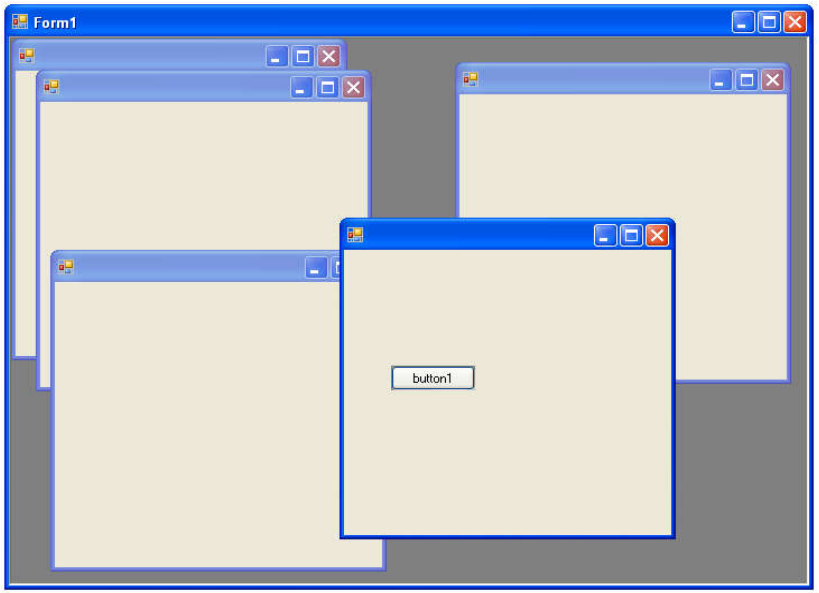
Código del Form2

- Agregar a la clase Form2 lo siguiente:

```
private:
 int presionado;
 System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
     {  presionado = 1;
     this->Close();
 }
 System::Void button2_Click(System::Object^ sender, System::EventArgs^ e)
     {  presionado = 2;
     this->Close();
 }
public:
 void ColocaNombre(String ^s)
 {
     label1->Text = s + " dime cual es tu color preferido";
 }
 String^ DameRespuesta()
 {
     if (presionado == 1)
         return "Rojo";
     else
         return "Verde";
 }
```

¿Que es MDI?

- MDI – Multiple Document Interface o Interfaz de múltiples documentos.
- Es un tipo de ventana o formulario que puede contener a otros formularios dentro. Los formularios dentro no pueden apoderarse del foco por lo que no sirve mostrarlos usando ShowDialog()



Como crear una aplicación MDI

- Para activar esta funcionalidad en un formulario, es necesario activar su propiedad IsMDIContainer.
- Por lo general el Form1 es el contenedor MDI por lo que a este es necesario configurarle la propiedad IsMDIContainer.
- El resto de formularios se pueden crear de forma dinámica de la misma forma como mostrado en la sección anterior.
- Sin embargo, es necesario especificarle a las nuevas ventanas que son hijas de una ventana MDI por lo que cuando las creemos dinámicamente deberíamos asignarle el padre MDI:

```
Form ^frm = gcnew Form();
frm->MdiParent = this;
frm->Show();
```

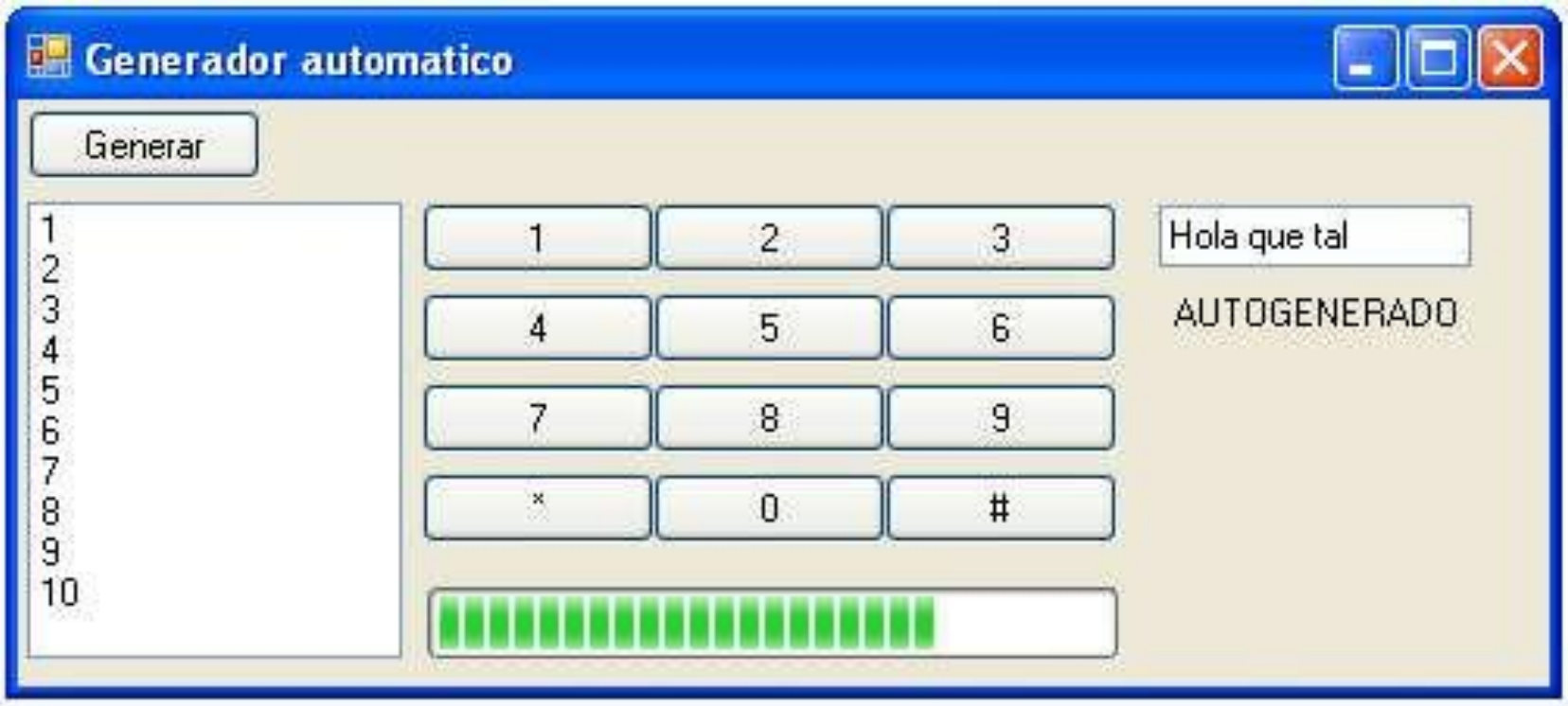
Opciones de un formulario MDI

Opción	Descripción
ActiveMdiChild	Devuelve un puntero a la ventana MDI que se encuentra activa. Ej. ActiveMdiChild->Text = “Hola”;
MdiChildren	Devuelve un arreglo .NET con los punteros a todas las ventanas MDI. Ejemplo – Cantidad de ventanas actuales int cant = MdiChildren->Length; Ejemplo – Cambiar el texto de la tercera ventana MdiChildren[2]->Text = “Esta es la ventana 3”;
LayoutMdi(MdiLayout)	Permite reorganizar todas las ventanas MDI. Recibe como parámetro el tipo de organización a implementar. Ejemplo LayoutMdi(MdiLayout::Cascade);

Ejercicios

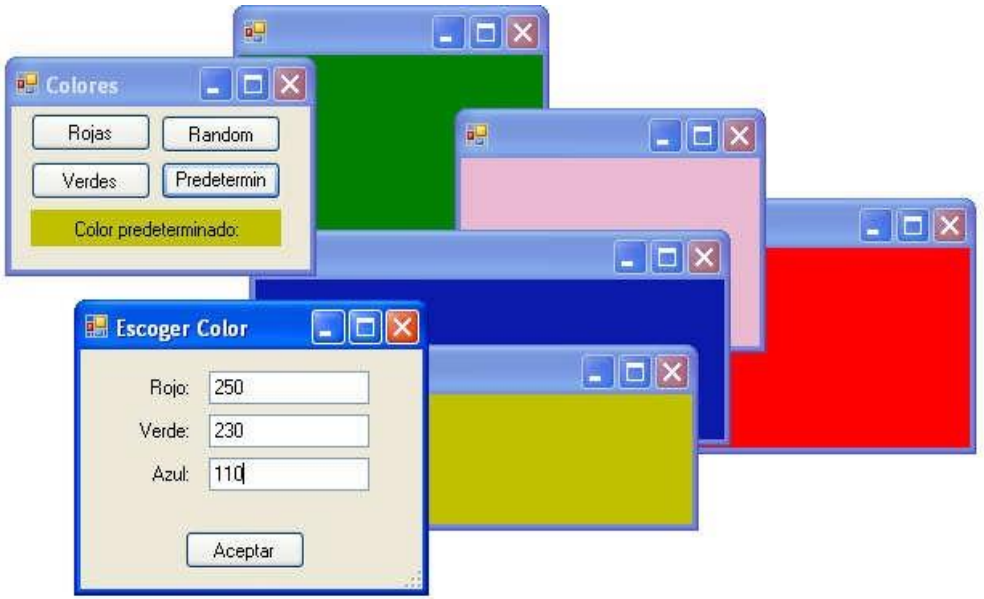
Ejercicio 1

- Crear un programa que al ejecutarse solo tenga un botón.
- Al presionar el botón deberá aparecer el siguiente formulario tal y como se muestra en la figura.



Ejercicio 2

- Crear un programa que tenga 4 botones, uno para crear formularios Rojos, otro para Verdes, otro para Random y uno para un color Predeterminado.
- Además deberá tener un botón que le permita escoger el color predeterminado desde una ventana aparte, tenga en cuenta que el color predeterminado deberá ser compuesto por rojo, verde y azul.



Ejercicio 3

- Crear un programa MDI que permita tener N ventanas. Cada ventana deberá mostrar uno de los programas utilizando canvas.

- Por ejemplo una ventana puede tener un rebote, la otra una calculadora, la siguiente un submarino, etc.

- Nota: Pueden existir ventanas con lo mismo.