	INSTITUTO TECNOLÓGICO DE NUEVO LAREDO **ING. EN SISTEMAS COMPUTACIONALES**			
	MATERIA: Programación Orientada a Objetos (C#)	**UNIDAD:** 5	**PRÁCTICA:** 2	
NOMBRE DE LA PRÁCTICA: Ejercicios aplicando delegados				
MAESTRO: Ing. Bruno López Takeyas, M.C.		**EMAIL: bruno.lt@nlaredo.tecnm.mx**		

OBJETIVO: El estudiante elaborará diagramas de clases en UML que apliquen delegados
MATERIAL Y EQUIPO NECESARIO: • Se recomienda la utilización de software para elaborar diagramas de clases de UML y diagramas de flujo • Elaborar programas de los ejercicios en C#

Lectura complementaria:
El método de ordenamiento de datos conocido como la burbuja.

Los algoritmos que ordenan un conjunto de datos se conocen como ordenadores, sorteadores o métodos de ordenamiento. Existe una gran variedad de ellos con diversas características y por ende, estrategias para ordenar los datos; sin embargo, en este momento nos concentraremos en el método de ordenamiento más sencillo de comprender y de implementar: la burbuja.

Este método tiene un arreglo de datos desordenados, lo recorre para comparar el contenido de sus celdas y de acuerdo a un criterio de ordenamiento (ascendente o descendente), los intercambia.

Existen dos criterios de ordenamiento de datos:

- **Ascendente:** Los datos se acomodan en secuencia del menor al mayor, es decir, cada dato sucesor debe ser mayor ó igual que su antecesor. En este caso $DATO_0 \leq DATO_1 \leq DATO_2 \leq \ldots. \leq DATO_n$.
- **Descendente:** Los datos se colocan sucesivamente del mayor al menor, o sea, cada dato sucesor debe ser menor ó igual que su antecesor. En este caso $DATO_0 \geq DATO_1 \geq DATO_2 \geq \ldots. \geq DATO_n$.

	INSTITUTO TECNOLÓGICO DE NUEVO LAREDO ING. EN SISTEMAS COMPUTACIONALES			
	MATERIA: Programación Orientada a Objetos (C#)	**UNIDAD:** 5	**PRÁCTICA:** 2	
NOMBRE DE LA PRÁCTICA: Ejercicios aplicando delegados				
MAESTRO: Ing. Bruno López Takeyas, M.C.		**EMAIL: bruno.lt@nlaredo.tecnm.mx**		

Intercambio de datos: Algunos métodos de ordenamiento realizan intercambios de datos en el arreglo como resultado de las comparaciones. Para ello, el método de ordenamiento implementa un método llamado `Intercambiar()` que recibe como parámetros el arreglo con los datos y los índices de las celdas cuyos valores serán intercambiados. Para realizar un intercambio de datos, es necesario utilizar una variable auxiliar del mismo tipo de dato que los valores en cuestión, siguiendo estos pasos:

1. *Copiar el valor del primer dato en la variable auxiliar.*
2. *Copiar el valor del segundo dato en la variable del primer dato.*
3. *Copiar el valor de la variable auxiliar en la variable del segundo dato.*

Enseguida se muestra el pseudocódigo del método para intercambiar datos, el cual será utilizado en el pseudocódigo de los métodos de ordenamiento (Fig. 1).

```
Intercambiar(Arreglo[], entero intCelda1, entero intCelda2): nulo

1. Auxiliar = Arreglo[intCelda1]
2. Arreglo[intCelda1] = Arreglo[intCelda2]
3. Arreglo[intCelda2] = Auxiliar
4. RETURN
```

Fig. 1.- Pseudocódigo del método que intercambia datos del arreglo.

El método de ordenamiento de la burbuja es probablemente el más sencillo y por ende, el más utilizado por estudiantes principiantes, sin embargo, también es el más ineficiente. Se conoce con este nombre debido a que durante el ordenamiento, el elemento más pequeño (ó el más

grande según el criterio de ordenamiento) se desplaza hacia la parte superior del arreglo, tal como lo hace una burbuja en el agua.

El principio fundamental de este método es comparar elementos adyacentes del arreglo y hacer los intercambios correspondientes de acuerdo al criterio de ordenamiento. Para ello se implementa un ciclo que controla la posición que ocupará el dato menor del arreglo (`i`). Este ciclo inicia en 1 y termina en la última celda del arreglo (`Arreglo.Tamaño-1`). Dentro de este ciclo se implementa otro ciclo que recorre el arreglo de derecha a izquierda para hacer las comparaciones e intercambios pertinentes (`j`). Este otro ciclo inicia en la última celda del arreglo (`Arreglo.Tamaño-1`) y se recorre de manera decreciente hasta llegar a la posición que ocupará el dato menor del arreglo (celda `i`).

Enseguida se muestra el pseudocódigo del método de la burbuja que recorre el arreglo de derecha a izquierda (Fig. 2).

```
Burbuja(Arreglo[]): nulo

1.- REPETIR CON i DESDE 1 HASTA Arreglo.Tamaño-1 CON INCREMENTO 1

    1.1. REPETIR CON j DESDE Arreglo.Tamaño-1 HASTA i CON DECREMENTO -1

        1.1.1 SI Arreglo[j] < Arreglo[j-1] ENTONCES
              1.1.1.1 Intercambiar(Arreglo, j, j-1)
         1.1.2. {FIN DE LA CONDICIONAL DEL PASO 1.1.1}

    1.2. {FIN DEL CICLO DEL PASO 1.1}
2.- {FIN DEL CICLO DEL PASO 1}
3.- RETURN
```

Fig. 2.- Pseudocódigo del método de la burbuja.

Al ejecutar la primera etapa del método, se realizan las comparaciones de los datos adyacentes de derecha a izquierda para hacer las comparaciones e intercambiarlos si es necesario. Para ello se utiliza un ciclo que inicia en la última celda y finaliza en la primera celda del arreglo donde se resaltan los movimientos realizados en el arreglo (Fig. 3).

Comparación	Datos	Intercambio	Arreglo después de la comparación
¿Arreglo[9] < Arreglo[8]?	¿0 < 4?	Si	0 1 2 3 4 5 6 7 8 9 8 6 7 5 1 3 2 9 **0** **4**
¿Arreglo[8] < Arreglo[7]?	¿0 < 9?	Si	0 1 2 3 4 5 6 7 8 9 8 6 7 5 1 3 2 **0** **9** 4
¿Arreglo[7] < Arreglo[6]?	¿0 < 2?	Si	0 1 2 3 4 5 6 7 8 9 8 6 7 5 1 3 **0** **2** 9 4
¿Arreglo[6] < Arreglo[5]?	¿0 < 3?	Si	0 1 2 3 4 5 6 7 8 9 8 6 7 5 1 **0** **3** 2 9 4
¿Arreglo[5] < Arreglo[4]?	¿0 < 1?	Si	0 1 2 3 4 5 6 7 8 9 8 6 7 5 **0** **1** 3 2 9 4
¿Arreglo[4] < Arreglo[3]?	¿0 < 5?	Si	0 1 2 3 4 5 6 7 8 9 8 6 7 **0** **5** 1 3 2 9 4

¿Arreglo[3] < Arreglo[2]?	¿0 < 7?	Si	0 1 2 3 4 5 6 7 8 9 8 6 0 7 5 1 3 2 9 4
¿Arreglo[2] < Arreglo[1]?	¿0 < 6?	Si	0 1 2 3 4 5 6 7 8 9 8 0 6 7 5 1 3 2 9 4
¿Arreglo[1] < Arreglo[0]?	¿0 < 8?	Si	0 1 2 3 4 5 6 7 8 9 0 8 6 7 5 1 3 2 9 4

Fig. 3.- Comparaciones e intercambios de la primera etapa del método de la burbuja izquierda.

	INSTITUTO TECNOLÓGICO DE NUEVO LAREDO **ING. EN SISTEMAS COMPUTACIONALES**			
	MATERIA: Programación Orientada a Objetos (C#)	**UNIDAD:** 5	**PRÁCTICA:** 2	
NOMBRE DE LA PRÁCTICA: Ejercicios aplicando delegados				
MAESTRO: Ing. Bruno López Takeyas, M.C.		**EMAIL: bruno.lt@nlaredo.tecnm.mx**		

Elabore el diagrama de clases en UML y la codificación de un programa para resolver los siguientes problemas:

1. Diseñe un proyecto de formas de Windows que capture los datos de objetos de estudiantes de una universidad, los almacene en un arreglo y los ordene de acuerdo a su matrícula. Los datos de cada estudiante son:
   - **Matrícula:** Cadena.
   - **Nombre:** Cadena.
   - **Grado:** Numérico entero.
   - **Grupo:** Caracter.
   - **Promedio:** Real.

Diseñe una clase llamada `Estudiante` con estos atributos y sus respectivas propiedades. Además esta clase debe implementar el método `CompareTo()` de la interfase `IComparable` para comparar objetos de estudiantes por su matrícula (Fig. 4).

	INSTITUTO TECNOLÓGICO DE NUEVO LAREDO **ING. EN SISTEMAS COMPUTACIONALES**			
	MATERIA: Programación Orientada a Objetos (C#)	**UNIDAD:** 5	**PRÁCTICA:** 2	
NOMBRE DE LA PRÁCTICA: Ejercicios aplicando delegados				
MAESTRO: Ing. Bruno López Takeyas, M.C.		**EMAIL: bruno.lt@nlaredo.tecnm.mx**		

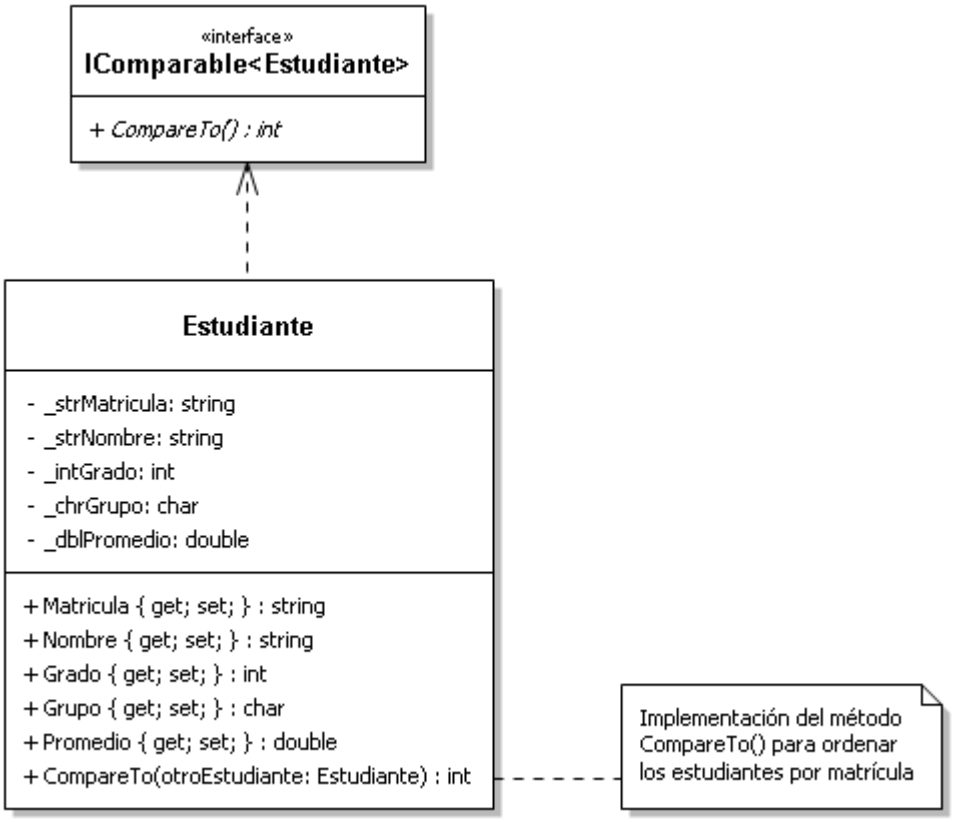

Fig. 4.- Diagrama de la clase Estudiante.

Diseñe una clase llamada `ClaseOrdenadores` que implemente el método de la burbuja para que reciba como parámetro el arreglo de estudiantes que desea ordenar así como un delegado `CriterioOrdenamiento` que determine la manera de ordenarlo (ascendente o descendente). Esta clase también contiene dos métodos estáticos (`Ascendente()` y `Descendente()`) que serán referenciados por el delegado y un método privado para realizar el intercambio de datos (`Intercambia()`) (Fig. 5).

	INSTITUTO TECNOLÓGICO DE NUEVO LAREDO ING. EN SISTEMAS COMPUTACIONALES			
	MATERIA: Programación Orientada a Objetos (C#)	**UNIDAD:** 5	**PRÁCTICA:** 2	
NOMBRE DE LA PRÁCTICA: Ejercicios aplicando delegados				
MAESTRO: Ing. Bruno López Takeyas, M.C.		**EMAIL: bruno.lt@nlaredo.tecnm.mx**		

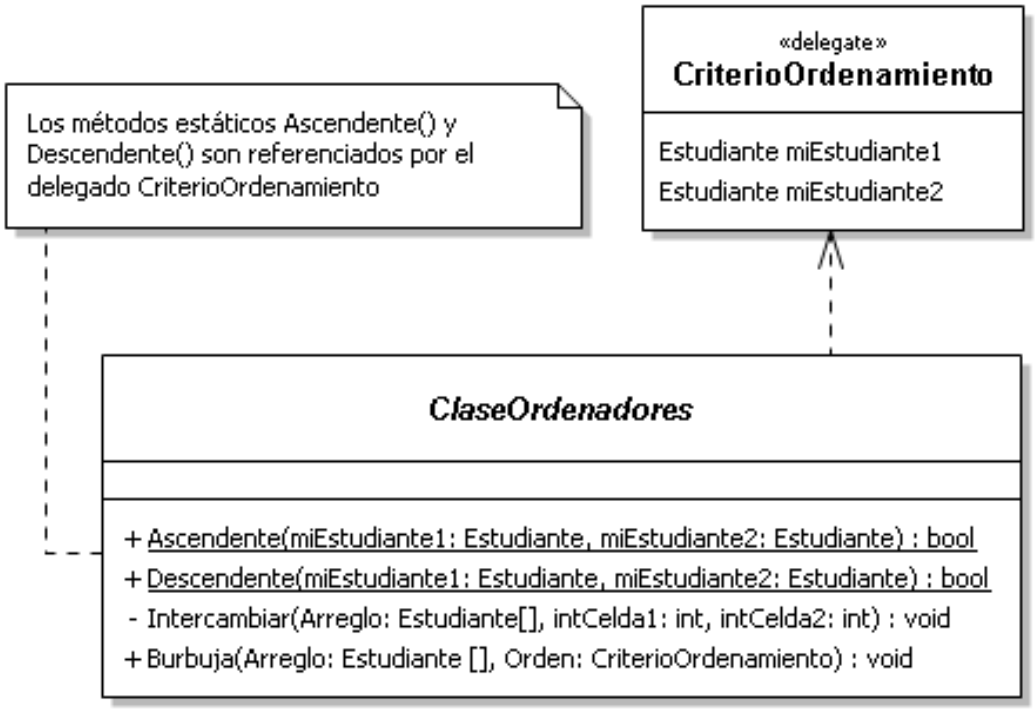

Fig. 5.- Diagrama de la ClaseOrdenadores.

	INSTITUTO TECNOLÓGICO DE NUEVO LAREDO **ING. EN SISTEMAS COMPUTACIONALES**			
	MATERIA: Programación Orientada a Objetos (C#)	**UNIDAD:** 5	**PRÁCTICA:** 2	
NOMBRE DE LA PRÁCTICA: Ejercicios aplicando delegados				
MAESTRO: Ing. Bruno López Takeyas, M.C.		**EMAIL: bruno.lt@nlaredo.tecnm.mx**		

Diseñe una forma para administrar y ordenar los datos de los estudiantes (Fig. 6).

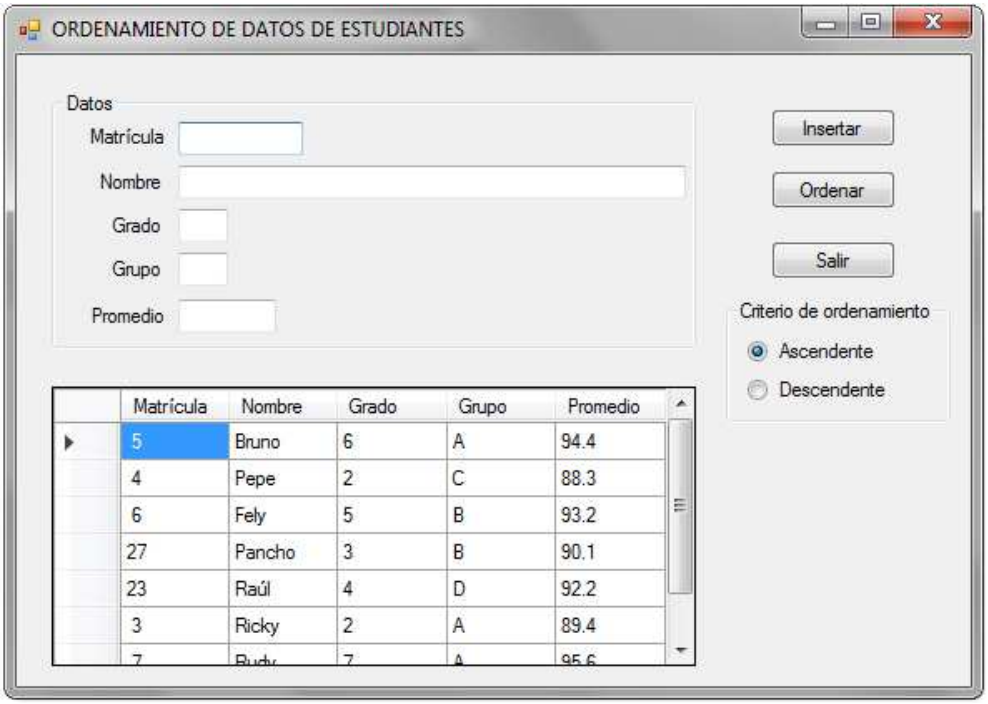

Fig. 6. Diseño de la forma que controla los datos de los estudiantes.

Para insertar los datos de un estudiante basta capturarlos en los *textBoxes* correspondientes y oprimir el botón Insertar. Al hacerlo, se agregan los datos capturados al *dataGridView*.

Para ordenar los datos de los estudiantes, primero debe seleccionarse el criterio de ordenamiento (ascendente o descendente), oprimiendo el *radioButton* correspondiente para después oprimir el botón Ordenar.

Cuando se desean ordenar los datos, se crea un arreglo local de objetos al que se insertan los datos recuperados del *dataGridView*, luego se envía este arreglo al método de ordenamiento y al terminar, se despliegan nuevamente los datos ordenados en el *dataGridView*.

Al invocar el método de la burbuja, debe enviársele como parámetros el arreglo que se desea ordenar y el delegado con el nombre del método que se requiere ejecutar de acuerdo al criterio de ordenamiento seleccionado por el usuario (Fig. 7).

```
if (radAscendente.Checked)
  ClaseOrdenadores.Burbuja(Arreglo, ClaseOrdenadores.Ascendente);
if (radDescendente.Checked)
  ClaseOrdenadores.Burbuja(Arreglo, ClaseOrdenadores.Descendente);
```

Fig. 7.- Ejecución del método de la burbuja.

2. Modifique el diseño y aplicación anteriores para que pueda ordenar los datos de los estudiantes por cualquier atributo (no solamente por matrícula). Para ello, agregue un *groupBox* con *radioButtons* a la forma para que el usuario seleccione el atributo por el cual se desea hacer el ordenamiento de los datos (Fig. 8).

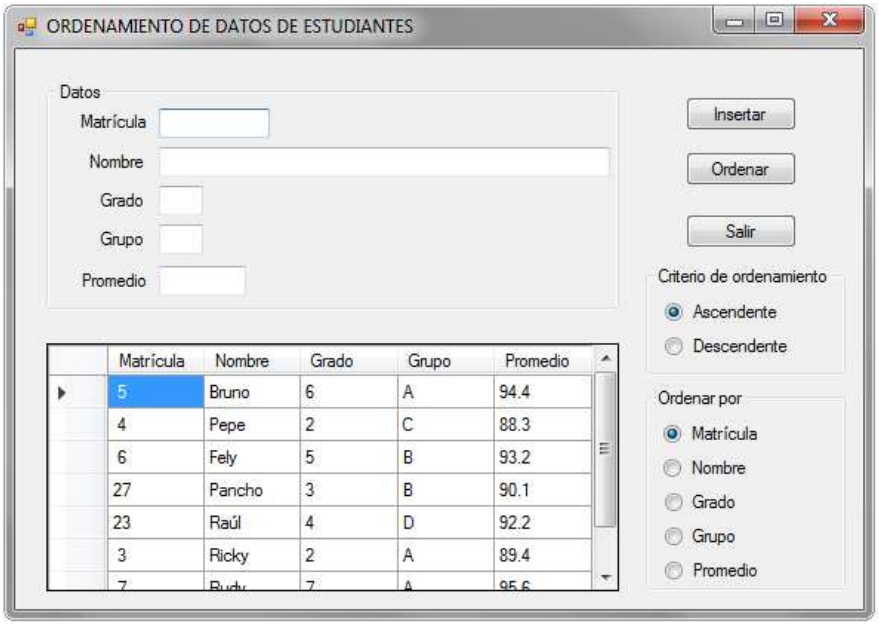

Fig. 8. Modificación de la forma para ordenar los datos por cualquier atributo.

	INSTITUTO TECNOLÓGICO DE NUEVO LAREDO ING. EN SISTEMAS COMPUTACIONALES			
	MATERIA: Programación Orientada a Objetos (C#)	**UNIDAD:** 5	**PRÁCTICA:** 2	
NOMBRE DE LA PRÁCTICA: Ejercicios aplicando delegados				
MAESTRO: Ing. Bruno López Takeyas, M.C.		**EMAIL: bruno.lt@nlaredo.tecnm.mx**		

Modifique la clase `Estudiante` y agréguele el delegado `AtributoComparable` y un método estático por cada atributo para ser invocado por él (Fig. 9).

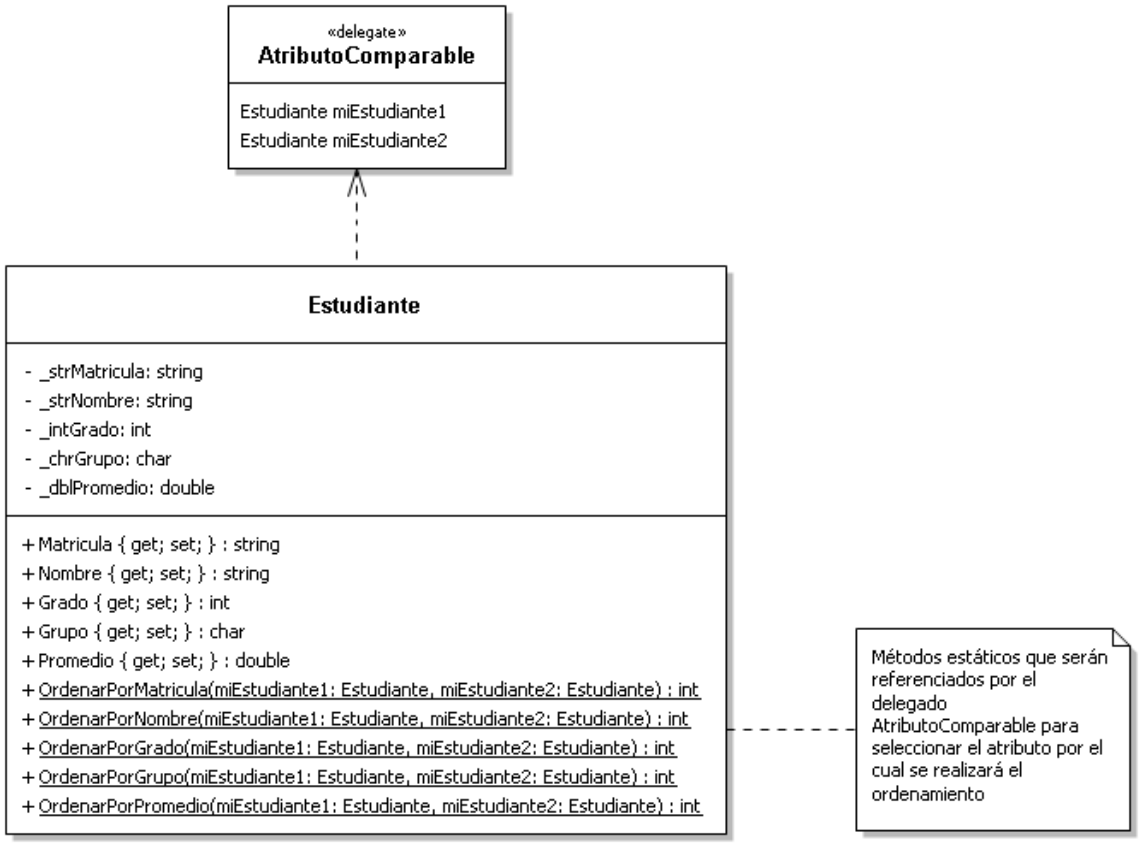

Fig. 9.- Diagrama de la clase Estudiante.

	INSTITUTO TECNOLÓGICO DE NUEVO LAREDO ING. EN SISTEMAS COMPUTACIONALES			
	MATERIA: Programación Orientada a Objetos (C#)	**UNIDAD:** 5	**PRÁCTICA:** 2	
NOMBRE DE LA PRÁCTICA: Ejercicios aplicando delegados				
MAESTRO: Ing. Bruno López Takeyas, M.C.		**EMAIL: bruno.lt@nlaredo.tecnm.mx**		

Modifique el delegado `CriterioOrdenamiento` de la clase `ClaseOrdenadores` para que ahora también reciba como parámetro el delegado `AtributoComparable` de la clase `Estudiante`. Al hacerlo, debe incluir este parámetro en los métodos estáticos `Ascendente()` y `Descendente()`. Con esto se logra que los datos de los estudiantes se ordenen ya sea de forma ascendente o descendente por medio del delegado `CriterioOrdenamiento` de la `ClaseOrdenadores` y también por cada uno de sus atributos por medio del delegado `AtributoComparable` de la clase `Estudiante` (Fig. 10).

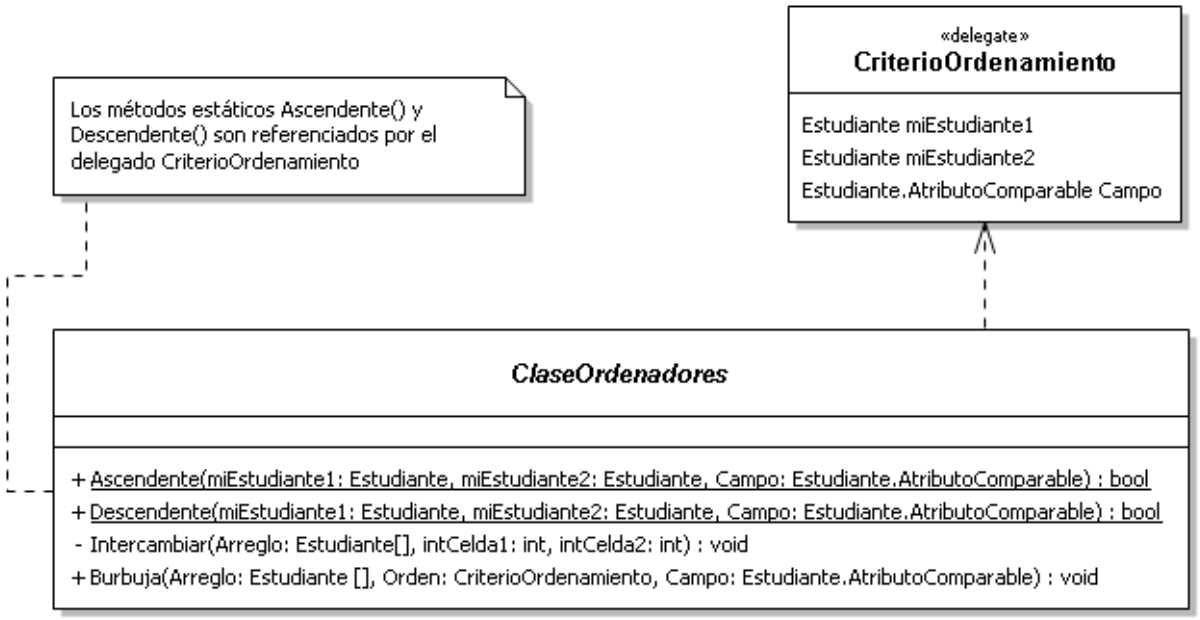

Fig. 10.- Diagrama de la ClaseOrdenadores.

3. Diseñe una clase para controlar el radio de una circunferencia y que solamente tenga un método que utilice un delegado para calcular tanto el área como su perímetro (Fig.11).

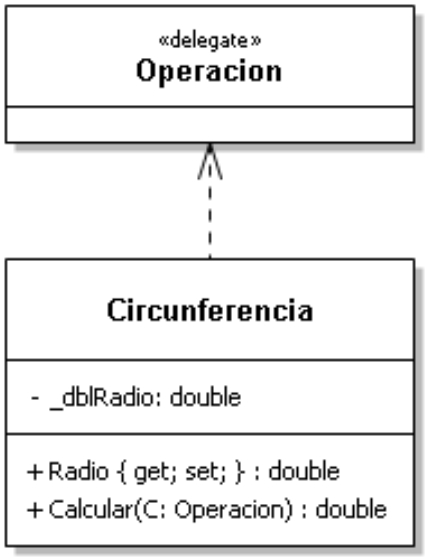

Fig. 11.- Diagrama de la clase Circunferencia.

Diseñe una forma que contenga un *textBox* para capturar el valor del radio, dos *radioButtons* para seleccionar el tipo de operación deseada y un botón para realizar el cálculo correspondiente (Fig. 12).

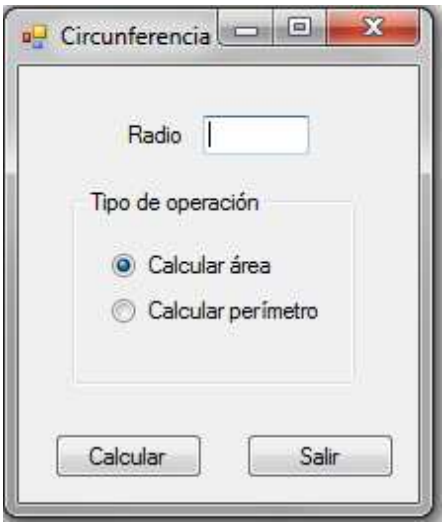

Fig. 12.- Diseño de la forma.

Cuando invoque el método `Calcular()`, envíe la implementación de la fórmula correspondiente por medio de una expresión lambda.

4. Diseñe un diagrama de clases donde establezca una relación de composición entre un pastel y sus ingredientes. Los datos del pastel son:
    - Nombre (cadena).
    - Lista de ingredientes que lo componen (`List<Ingrediente> ListaIngredientes`).

    Mientras que sus métodos son:
    - *Agregar ingrediente.-* Inserta un objeto de tipo `Ingrediente` a la lista de ingredientes.
    - *Eliminar ingrediente.-* Elimina un objeto de tipo `Ingrediente` de la lista de ingredientes.
    - *Destructor de la clase.-* Elimina la lista de ingredientes cuando se destruye un objeto del pastel.
    - *Iterador* `GetEnumerator().` – Sirve para recorrer la lista de ingredientes para mostrarlos en pantalla.
    - *Sobreescritura del método* `ToString().` – Se utiliza para mostrar en pantalla el nombre de un pastel.
    - *Ordenar ingredientes.-* Este método utiliza un delegado para ordenar los ingredientes de acuerdo a su nombre tanto en forma ascendente como descendente.
    - *Ascendente.-* Este método es invocado por el método `OrdenarIngredientes()` para ordenar los ingredientes por nombre de manera ascendente.

- *Descendente.-* Este método es invocado por el método `OrdenarIngredientes()` para ordenar los ingredientes por nombre de manera descendente.
- *Delegado CriterioOrdenamiento.-* Se utiliza este delegado para invocar alguno de los métodos `Ascendente()` o `Descendente()` al ordenar la lista de ingredientes según se requiera.

Los datos de cada ingrediente son:

- Nombre (cadena).
- Cantidad a utilizar en el pastel (numérico real)

Mientras que sus métodos son:

- *Sobreescritura del método `ToString()`.* – Se utiliza para mostrar en pantalla los datos de un ingrediente.
- *Implementación del método `Equals()` de la interfase `IEquatable`.-* Se utiliza este método para localizar un ingrediente al momento de eliminarlo. Este método es requerido por el método `Remove()` de la colección genérica `List` y es utilizado a través del método `EliminarIngrediente()` de la `ListaIngredientes`.
- *Implementación del método `CompareTo()` de la interfase `IComparable`.-* Se utiliza este método para comparar ingredientes al momento de ordenarlos. Este método es requerido por el método `Sort()` de la colección genérica `List` y es utilizado por el método `OrdenarIngredientes()` a través del

método `Ascendente()` invocado por el delegado `CriterioOrdenamiento`.

La Fig. 13 muestra el diagrama de clases de la relación de composición entre el pastel y sus ingredientes.

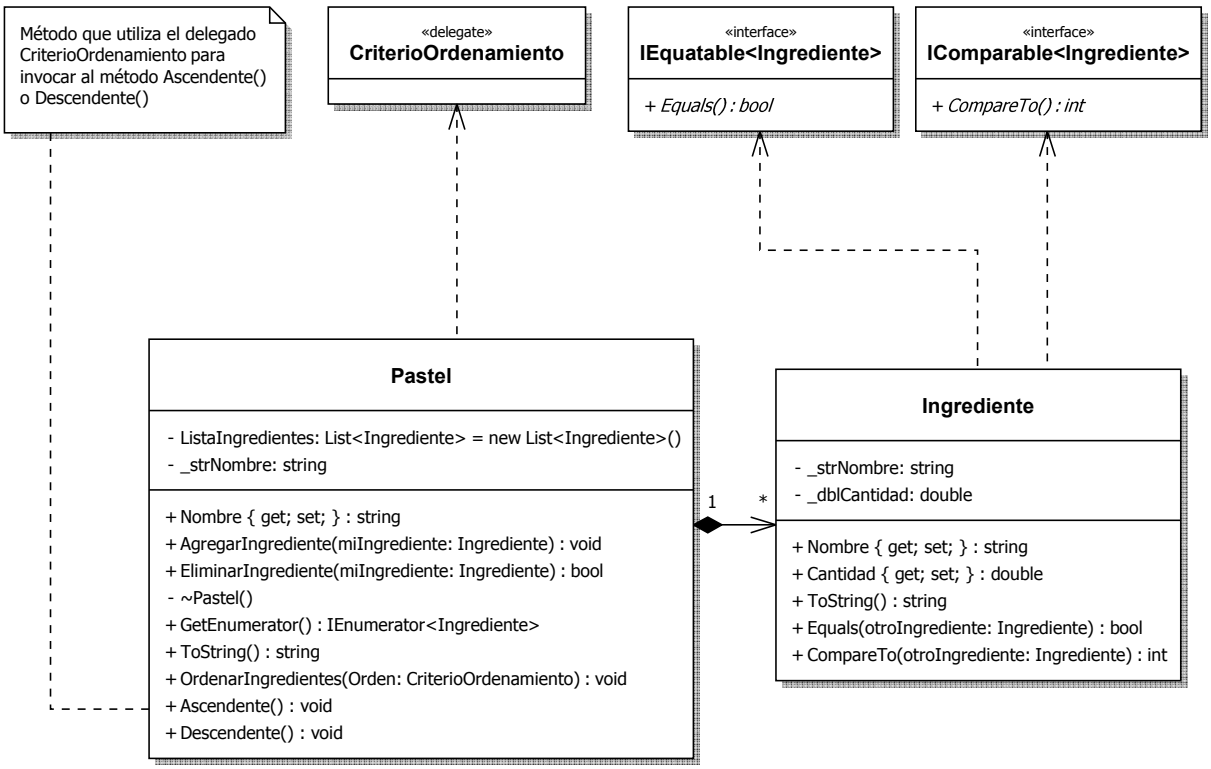

Fig. 13.- Diagrama de clases de la composición del pastel y sus ingredientes.

	INSTITUTO TECNOLÓGICO DE NUEVO LAREDO ING. EN SISTEMAS COMPUTACIONALES			
	MATERIA: Programación Orientada a Objetos (C#)	**UNIDAD:** 5	**PRÁCTICA:** 2	
NOMBRE DE LA PRÁCTICA: Ejercicios aplicando delegados				
MAESTRO: Ing. Bruno López Takeyas, M.C.		**EMAIL: bruno.lt@nlaredo.tecnm.mx**		

Diseñe la siguiente forma para operar esta composición y ordenar los datos de los ingredientes. En ella se destaca la presencia de dos *radioButtons* donde el usuario selecciona el criterio de ordenamiento deseado (Fig. 14).

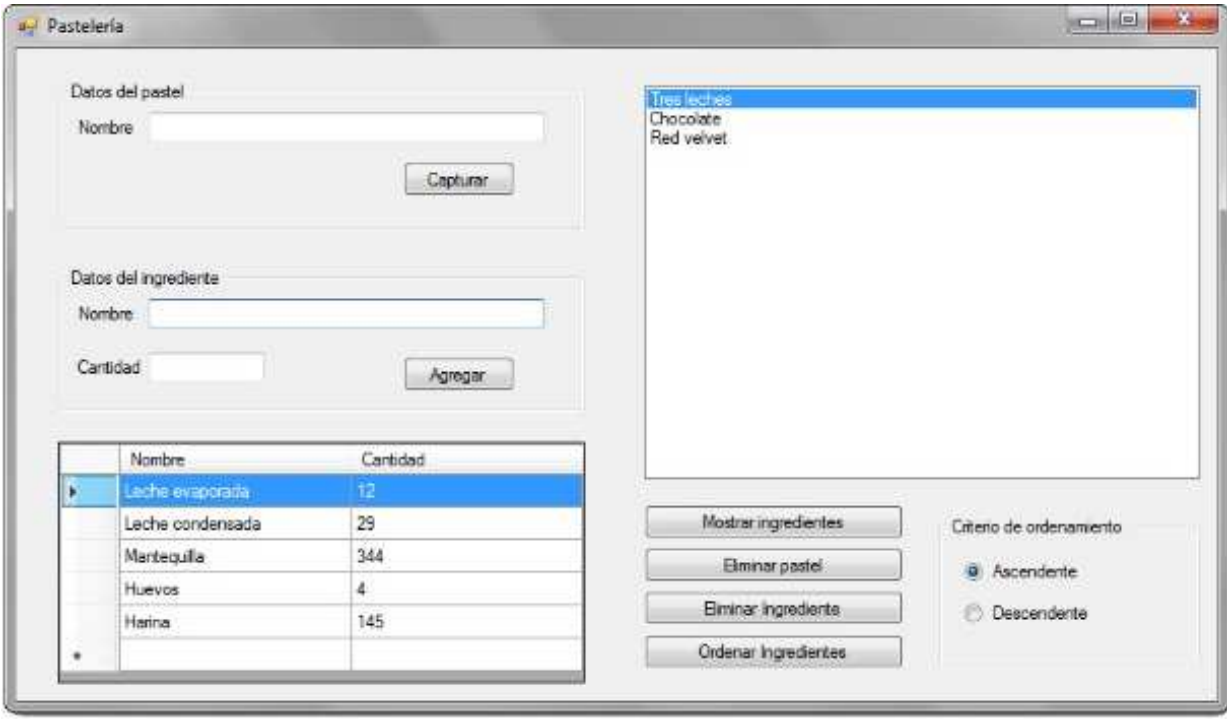

Fig. 14.- Diseño de la forma de la aplicación de la composición del pastel y sus ingredientes.

La clase `Pastel` tiene un delegado `CriterioOrdenamiento` que servirá para que el método `OrdenarIngredientes()` invoque ya sea al método `Ascendente()` o `Descendente()` (Fig. 15).

```
// Delegado
public delegate void CriterioOrdenamiento();

// Método para ordenar los ingredientes
public void OrdenarIngredientes(CriterioOrdenamiento Orden)
{
    Orden();
}
```

Fig. 15.- Definición del delegado y el método para ordenar los ingredientes.

El método `Ascendente()` de la clase `Pastel` invoca directamente el método `Sort()` de la `ListaIngredientes`, el cual a su vez, ejecuta el método `CompareTo()` de la clase Ingrediente para comparar dos ingredientes y determinar su ordenamiento (Fig. 16).

```
public void Ascendente()
      {
          // LLamada del método Sort() que utiliza el comparador
          // por default definido en la implementación del
          // método CompareTo() de la clase Ingrediente

          ListaIngredientes.Sort();
      }
```

Fig. 16.- Método que ordena ingredientes de forma ascendente.

El método `Descendente()` de la clase `Pastel` implementa una sobreescritura del método `Sort()` de la `ListaIngredientes` en la que utiliza un método anónimo y un delegado para invertir la comparación (ahora compara `miIngrediente2` con `miIngrediente1`, ya que antes era de manera inversa) y determinar el criterio de ordenamiento descendente (Fig. 17).

```
public void Descendente()
{
   // Sobrecarga del método Sort() que usa un método anónimo
   // para el delegado Comparison

  ListaIngredientes.Sort(delegate(Ingrediente miIngrediente1,
Ingrediente miIngrediente2)
  {
     return (miIngrediente2.CompareTo(miIngrediente1));
  }
  );
}
```

Fig. 17.- Método que sobreescribe el método Sort() para ordenar descendente.

5. Modifique la aplicación visual del ejercicio anterior y agregue otro delegado para poder ordenar los ingredientes de un pastel tanto por nombre como por cantidad; para ello, coloque *radioButtons* a la forma donde el usuario seleccione el campo por el que desea realizar el ordenamiento. NOTA: NO elimine el delegado y *radioButtons* del criterio de ordenamiento (ascendente o descendente).