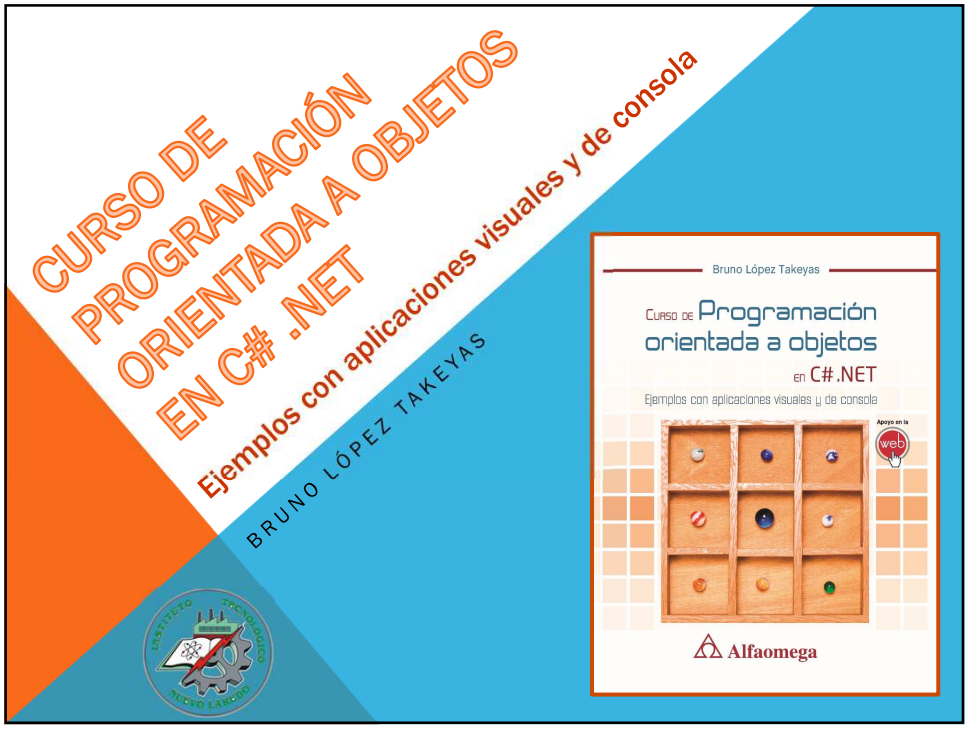

CURSO DE PROGRAMACIÓN ORIENTADA A OBJETOS EN C# .NET
Ejemplos con aplicaciones visuales y de consola
BRUNO LÓPEZ TAKEYAS
Bruno López Takeyas
Curso de Programación orientada a objetos en C#.NET
Ejemplos con aplicaciones visuales y de consola
Apoyo en la web
Alfaomega

1


CAPITULO 8
MÉTODOS DE ORDENAMIENTO CON DELEGADOS
3 1 4 1 5 9 2 6 5 4
i = 10
1 3 1 4 5 2 6 5 4 9
i = 9
1 1 3 4 2 5 5 4 6 9
i = 8
1 1 3 2 4 5 4 5 6 9
i = 7
1 1 2 3 4 4 5 5 6 9
i = 6
1 1 2 3 4 4 5 5 6 9
i = 5
1 1 2 3 4 4 5 5 6 9
i = 4
1 1 2 3 4 4 5 5 6 9
i = 3
1 1 2 3 4 4 5 5 6 9
i = 2
1 1 2 3 4 4 5 5 6 9
2

2

Métodos de ordenamiento

- **Introducción**
- **Criterios de ordenamiento**
- **Tipos de ordenamiento**
- **Métodos de ordenamiento interno**

Objetivo

- Conocer un algoritmo para ordenar datos almacenados en un arreglo de tal forma que puedan ser fácilmente procesados en aplicaciones para resolver problemas cotidianos.

3

Introducción

- Ordenar un conjunto de datos almacenados en un arreglo
- El ordenamiento de datos es una actividad que consiste en reorganizarlos o recolocarlos en un lugar determinado para que cumplan con una secuencia específica.
- Al ordenar los datos, se pretende que su ubicación, acceso y uso sea lo más rápido posible y esto se logra de acuerdo al criterio de ordenamiento.

4

Criterios de ordenamiento

- **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 $DATO_0 \leq DATO_1 \leq DATO_2 \leq .... \leq DATO_n$.

- **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 $DATO_0 \geq DATO_1 \geq DATO_2 \geq .... \geq DATO_n$.

5

Tipos de ordenamiento

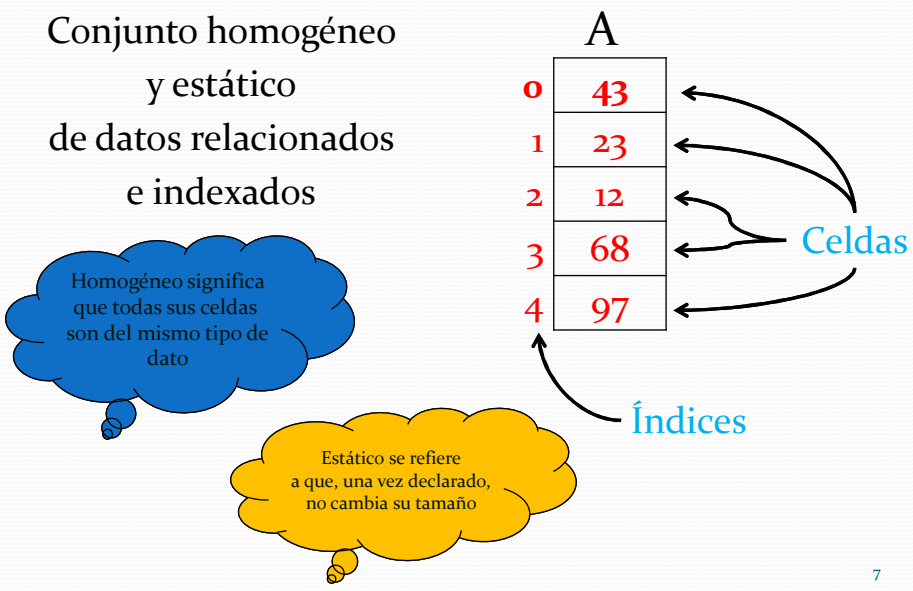
- **Ordenamiento interno:** Este tipo de ordenamiento se aplica cuando los elementos a ordenar se encuentran almacenados en alguna estructura de datos ubicada en la memoria principal. Típicamente se utilizan en arreglos, sin embargo, pueden aplicarse a cualquiera de las estructuras de datos analizadas previamente.

- **Ordenamiento externo:** Este se aplica cuando los datos están almacenados en archivos que se ubican en dispositivos de almacenamiento secundario como discos duros, cintas, memorias, etc.

6

Arreglo

Conjunto homogéneo
y estático
de datos relacionados
e indexados



7

7

Declaración de arreglos

- No sólo basta con declararlo, sino también debe crearse con el operador **new**

```
int [ ]  arreglo;   // declara el arreglo
arreglo = new int[12];  // reserva memoria

double [ ] arreglo2 = new double[10];
```

8

8

Manejo del tamaño del arreglo

- Declarar una constante con el tamaño del arreglo

```
const int Tamaño= 15;
int [ ]  arreglo;   // declara el arreglo
arreglo = new int[Tamaño];
// reserva memoria
```

9

9

Arreglos “dinámicos”

- Se pueden crear arreglos cuyo tamaño se pueda establecer dinámicamente a partir de una expresión que produzca un valor entero

```
int x = 15;  // variable
int [ ]  arreglo;   // declara el arreglo
arreglo = new int[x];  // reserva memoria
```

10

10

Recorrido de arreglos

```
int [ ] a = { 1, 2, 3, 4, 5, 6 };

for(int i=0; i<= a.Length; i++)
  Console.Write(“\n{0}”, a[i]);
```

- La longitud de un arreglo se obtiene con la expresión **Length**

11

11

Recorrido de un arreglo con ciclo *foreach*

```
string[] Alumno= new string[]{"Pepe",
“Rodolfo", "Maria", “Fabiola",
"Miguel"};

foreach(string nombre in Alumno)
{
 Console.WriteLine("\n{0}",nombre);
}
```

12

12

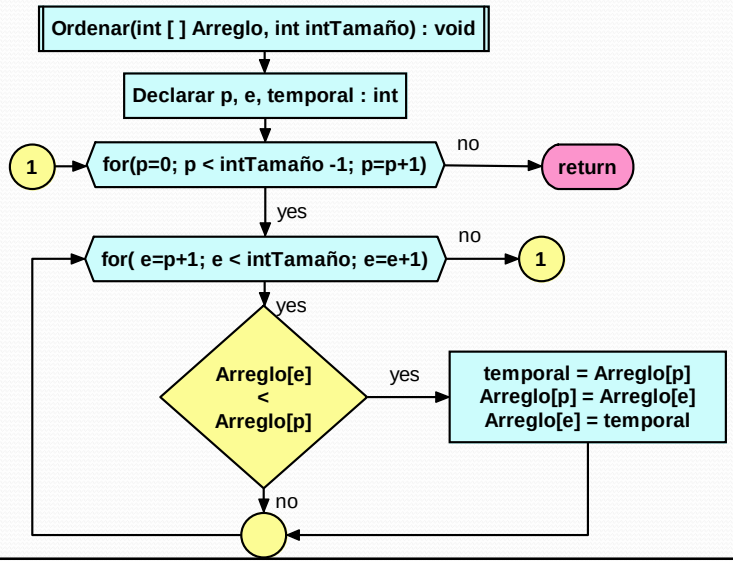
Intercambio de datos de un arreglo

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.*

```
    Intercambia(Arreglo[], entero a, entero b): nulo

1. Auxiliar = Arreglo[a]
2. Arreglo[a] = Arreglo[b]
3. Arreglo[b] = Auxiliar
4. RETURN
```

13

Diagrama de flujo de un método para ordenar un arreglo de números enteros



14

14

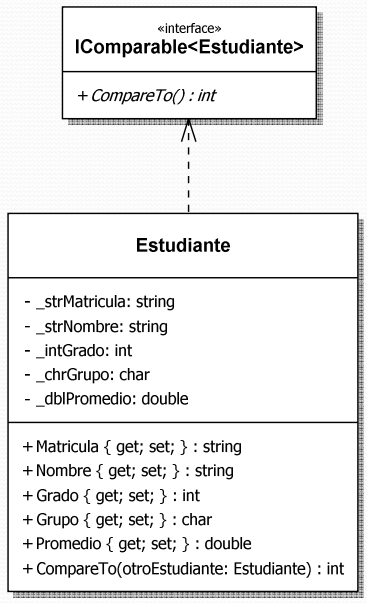
Diseño de una aplicación sencilla (sin delegados)

- Ordenar datos de estudiantes
- Datos de los estudiantes
  - *Matrícula (string)*
  - *Nombre (string)*
  - *Grado (int)*
  - *Grupo (char)*
  - *Promedio (double)*
- Ordenar los objetos de manera ascendente por promedio

15

15

Definición de la clase con los datos



16

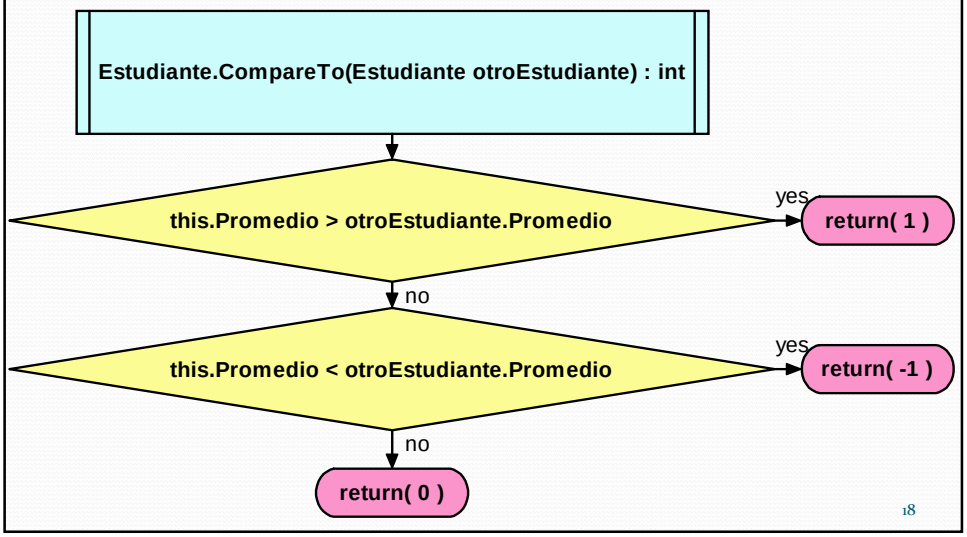
Arreglo de objetos de tipo Estudiante

```
const int Tamaño = 10;

Estudiante [] miArreglo;
miArreglo = new Estudiante[Tamaño];
```

17

17

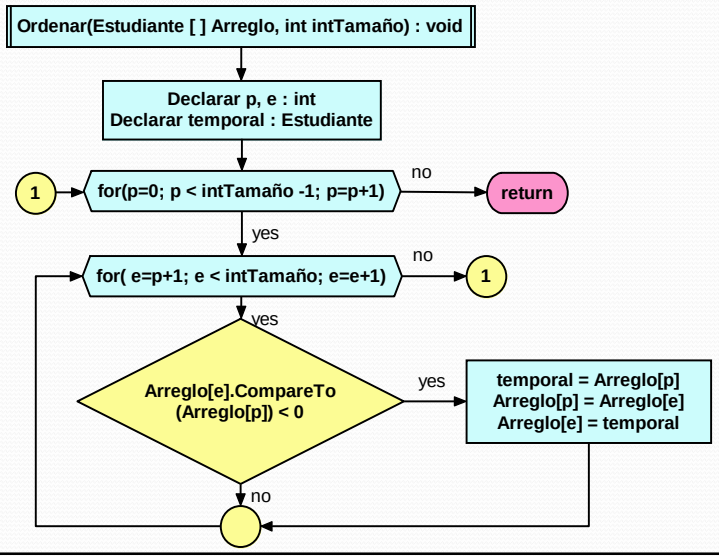
Diagrama de flujo del método de comparación CompareTo( )



18

18

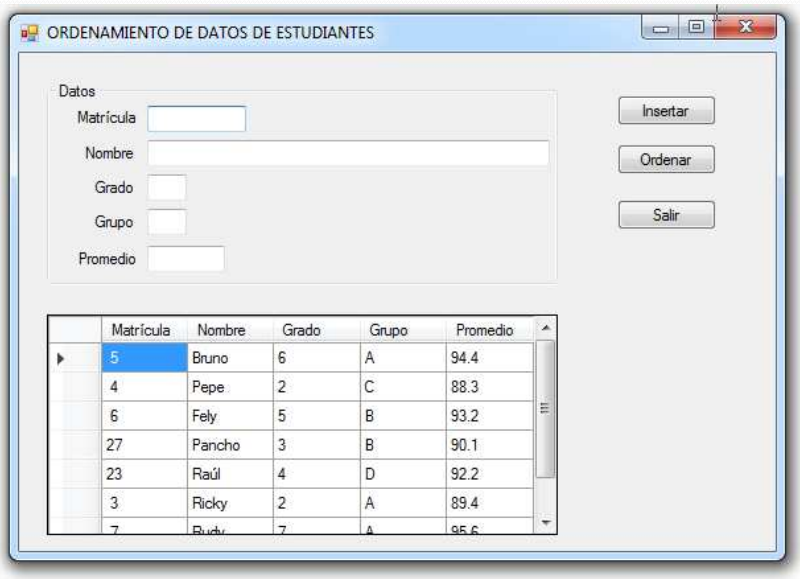
Diagrama de flujo del método para ordenar un arreglo de objetos de tipo Estudiante



19

19

Diseño de la forma



20

20

Características de la aplicación

- Ordena datos de estudiantes
- Criterio de ordenamiento: Ascendente
- Considera el campo del **promedio** para hacer las comparaciones de los objetos

- ¿Cómo seleccionar el criterio de ordenamiento? Ascendente o descendente
- ¿Cómo seleccionar el campo a utilizar para comparar los objetos?

21

21

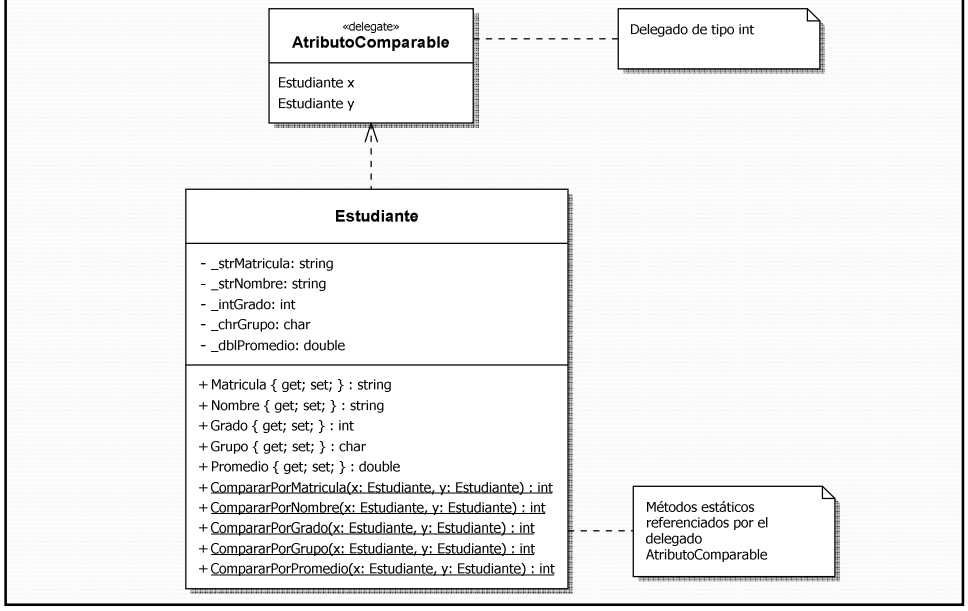
Diseño de una aplicación con delegados

- Ordenar datos de estudiantes
- Seleccionar el criterio de ordenamiento
  - *Ascendente*
  - *Descendente*
- Seleccionar el campo para hacer las comparaciones
  - *Matrícula (string)*
  - *Nombre (string)*
  - *Grado (int)*
  - *Grupo (char)*
  - *Promedio (double)*

22

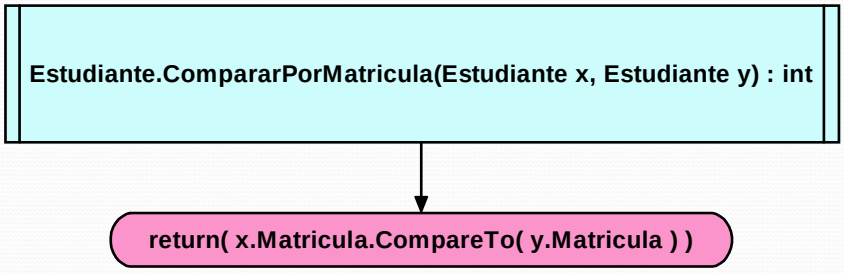
22

Definición de la clase con los datos



23

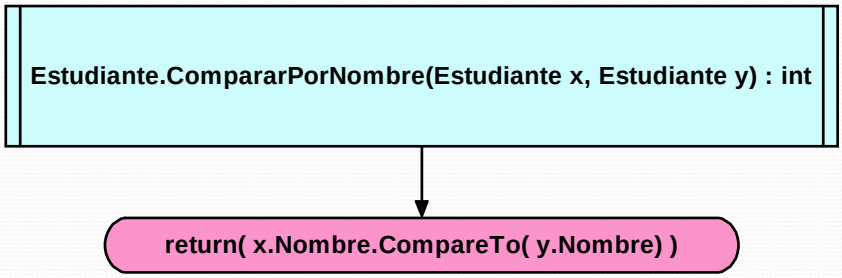
Diagrama de flujo del método de comparación para ordenar por matrícula



24

24

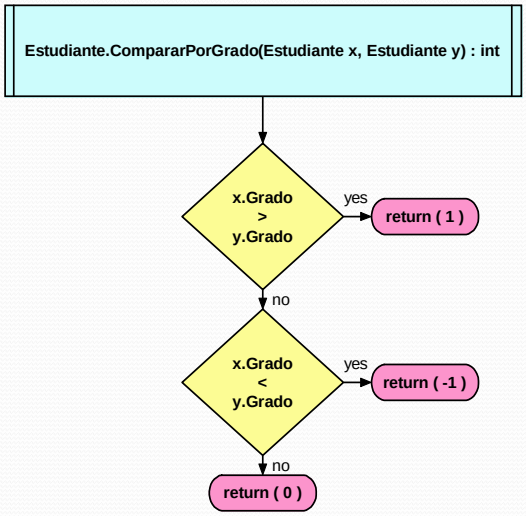
Diagrama de flujo del método de comparación para ordenar por nombre



25

25

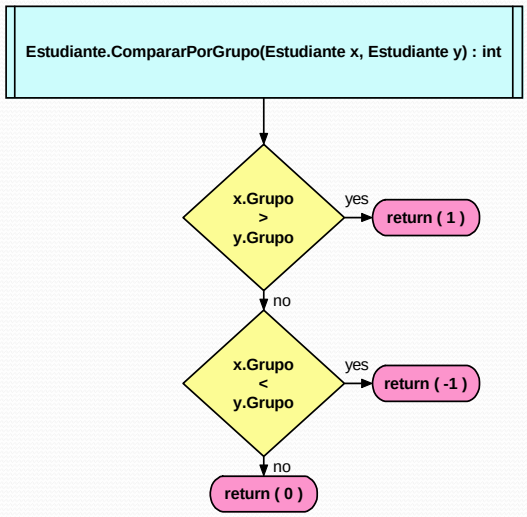
Diagrama de flujo del método de comparación para ordenar por grado



26

26

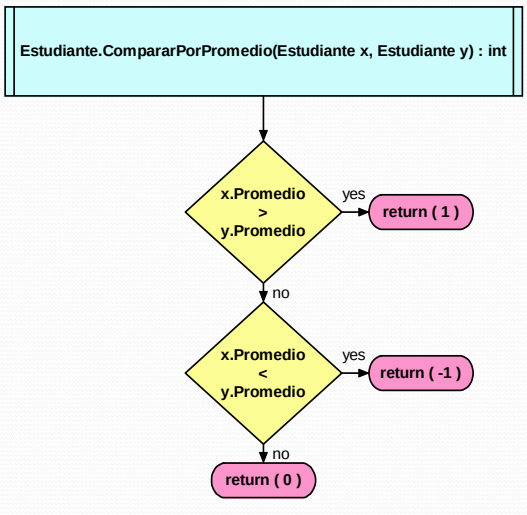
Diagrama de flujo del método de comparación para ordenar por grupo



27

27

Diagrama de flujo del método de comparación para ordenar por promedio



28

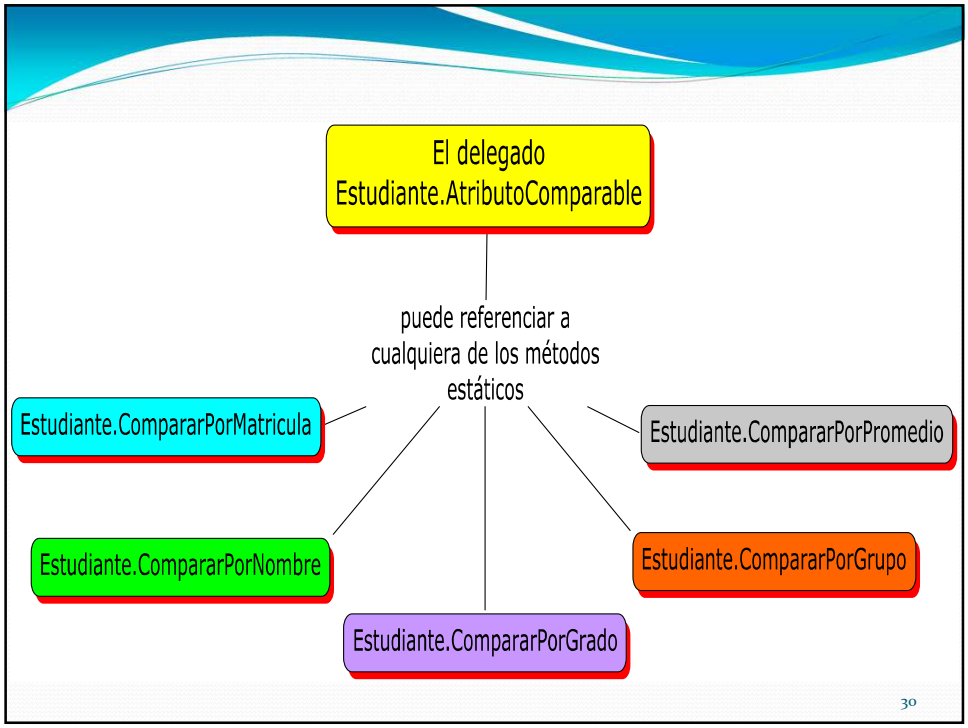
28

El delegado AtributoComparable

- Permite declarar una variable como:
  - `Estudiante.AtributoComparable campo;`
- La variable `campo` representa un método que puede contener la referencia a cualquiera de los siguientes métodos estáticos de la clase `Estudiante`:
  - `Estudiante.CompararPorMatricula`
  - `Estudiante.CompararPorNombre`
  - `Estudiante.CompararPorGrado`
  - `Estudiante.CompararPorGrupo`
  - `Estudiante.CompararPorPromedio`

29

29



30

El delegado AtributoComparable (*cont.*)

- Recibe 2 parámetros que representan los objetos de los estudiantes que se desean comparar:
  - `Estudiante x`
  - `Estudiante y`
- Esos objetos están ubicados en el arreglo en las celdas `p` y `e` respectivamente.

31

31

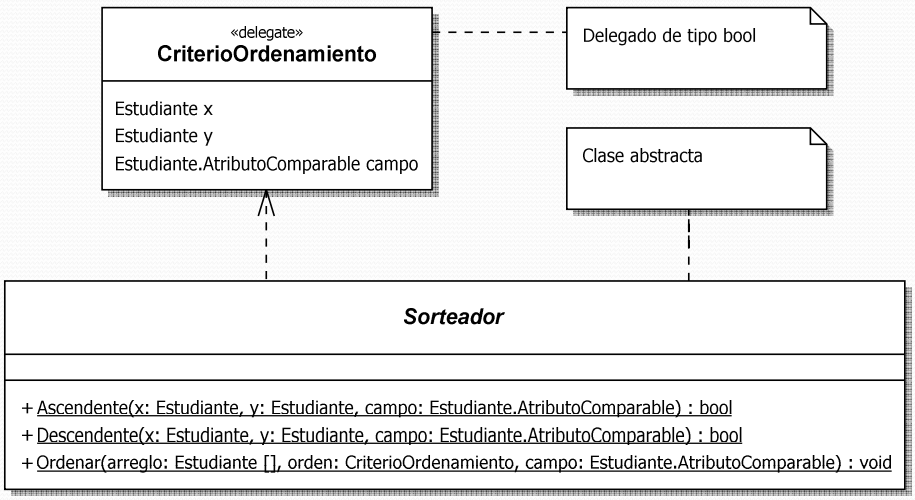
El delegado AtributoComparable (*cont.*)

- Devuelve un valor de tipo entero:
  - ***1*** *cuando el* `Estudiante x` *es mayor que el* `Estudiante y`
  - ***-1*** *cuando el* `Estudiante x` *es menor que el* `Estudiante y`
  - ***0*** *cuando el* `Estudiante x` *es igual que el* `Estudiante y`

32

32

Definición de la clase con el método de ordenamiento



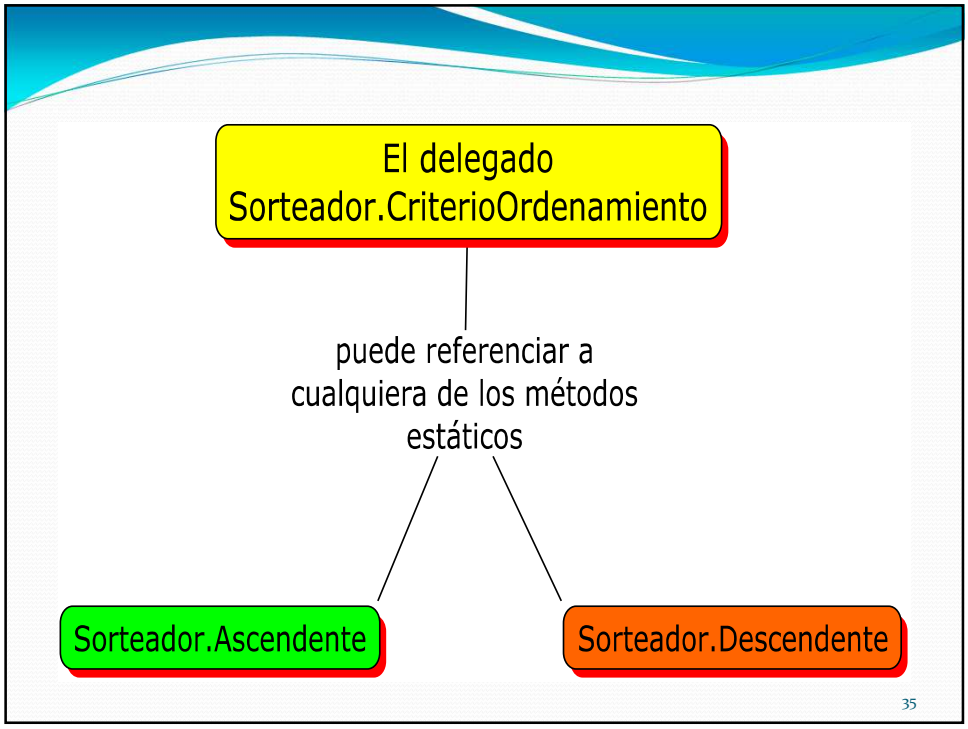
33

El delegado CriterioOrdenamiento

- Permite declarar una variable como:
  - `Sorteador.CriterioOrdenamiento orden;`
- La variable `orden` representa un método que puede contener la referencia a cualquiera de los siguientes métodos estáticos de la clase `Sorteador`:
  - `Sorteador.Ascendente`
  - `Sorteador.Descendente`

34

34



35

El delegado CriterioOrdenamiento (*cont.*)

- Recibe 3 parámetros que representan los objetos de los estudiantes que se desean comparar y el método de comparación:
  - `Estudiante x`
  - `Estudiante y`
  - `Estudiante.AtributoComparable campo`
- Esos objetos están ubicados en el arreglo en las celdas `p` y `e` respectivamente.

36

36

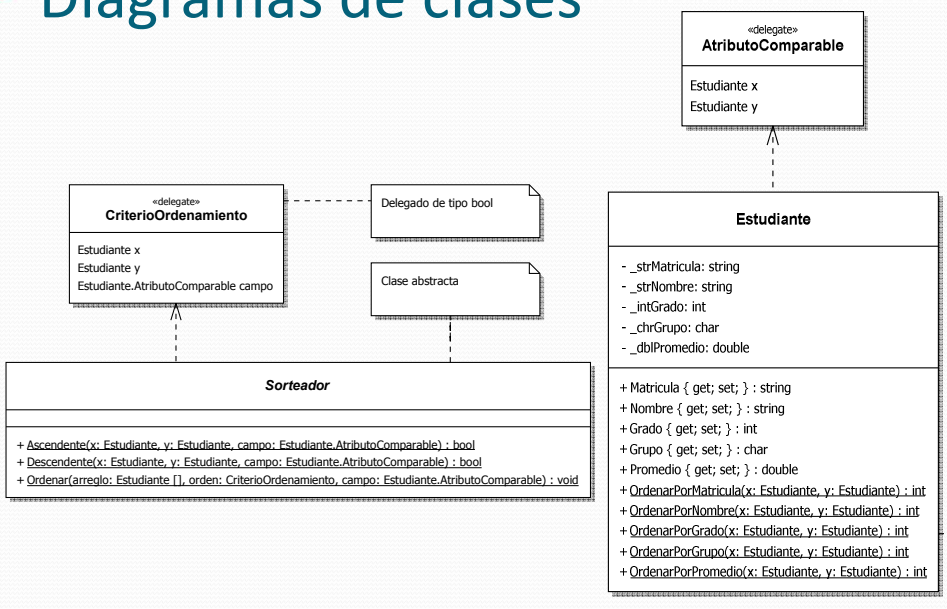
El delegado CriterioOrdenamiento (*cont.*)

- Devuelve un valor de tipo booleano:
  - ***true*** *cuando se cumple la condición de la comparación y se intercambiarán los objetos* `Estudiante x` *y* `Estudiante y` *en el arreglo*
  - ***false*** *en caso contrario*

37

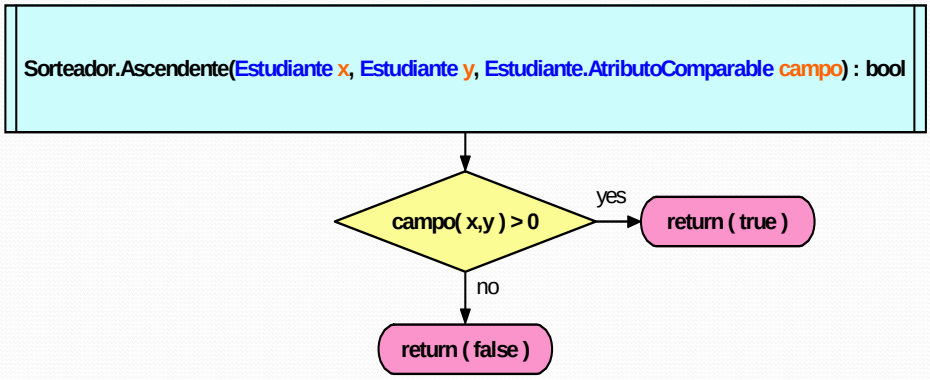
37

Diagramas de clases



38

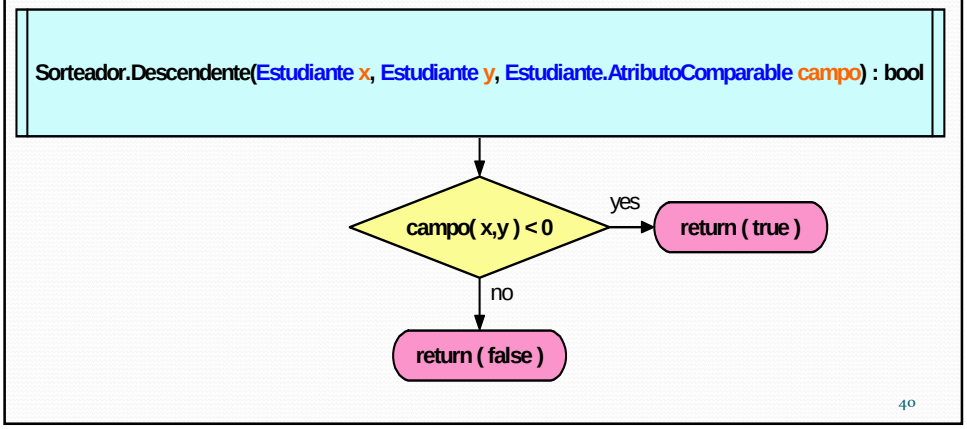
Diagrama de flujo del método para el criterio de ordenamiento ascendente



39

39

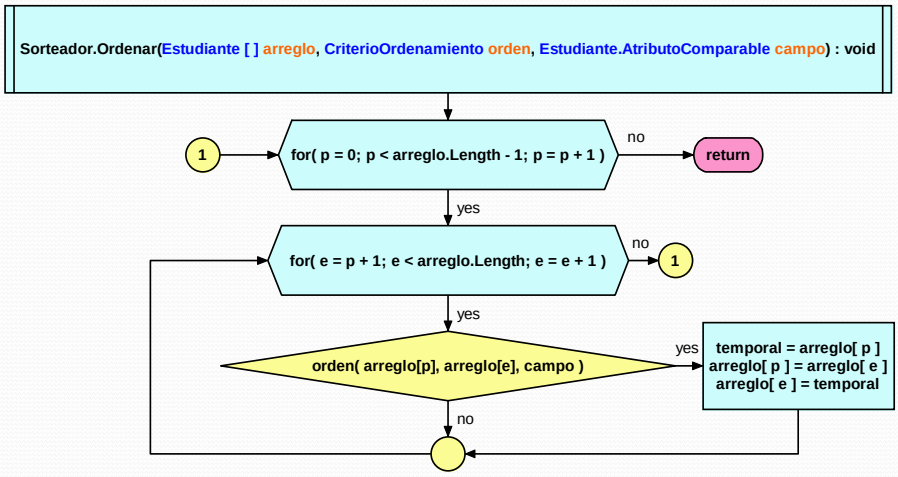
Diagrama de flujo del método para el criterio de ordenamiento descendente



40

40

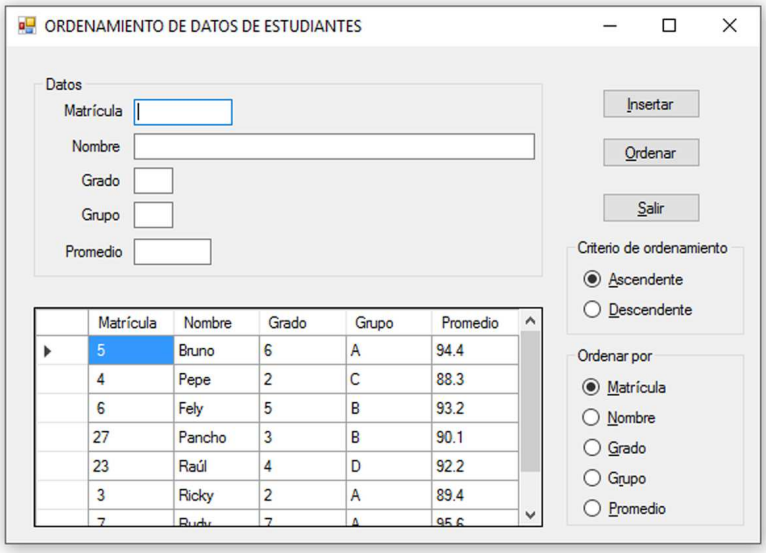
Diagrama de flujo del método de ordenamiento con delegados



41

41

Diseño de la forma



42

42

Codificación

```
if (radAscendente.Checked && radPorMatricula.Checked)
  Sorteador.Ordenar(arreglo, Sorteador.Ascendente, Estudiante.CompararPorMatricula);

if (radAscendente.Checked && radPorNombre.Checked)
  Sorteador.Ordenar(arreglo, Sorteador.Ascendente, Estudiante.CompararPorNombre);

if (radAscendente.Checked && radPorGrado.Checked)
  Sorteador.Ordenar(arreglo, Sorteador.Ascendente, Estudiante.CompararPorGrado);

if (radAscendente.Checked && radPorGrupo.Checked)
  Sorteador.Ordenar(arreglo, Sorteador.Ascendente, Estudiante.CompararPorGrupo);

if (radAscendente.Checked && radPorPromedio.Checked)
  Sorteador.Ordenar(arreglo, Sorteador.Ascendente, Estudiante.CompararPorPromedio);

...
...
```

43

43

Codificación (*cont.*)

```
if (radDescendente.Checked && radPorMatricula.Checked)
   Sorteador.Ordenar(arreglo, Sorteador.Descendente, Estudiante.CompararPorMatricula);

if (radDescendente.Checked && radPorNombre.Checked)
   Sorteador.Ordenar(arreglo, Sorteador.Descendente, Estudiante.CompararPorNombre);

if (radDescendente.Checked && radPorGrado.Checked)
   Sorteador.Ordenar(arreglo, Sorteador.Descendente, Estudiante.CompararPorGrado);

if (radDescendente.Checked && radPorGrupo.Checked)
   Sorteador.Ordenar(arreglo, Sorteador.Descendente, Estudiante.CompararPorGrupo);

if (radDescendente.Checked && radPorPromedio.Checked)
   Sorteador.Ordenar(arreglo, Sorteador.Descendente, Estudiante.CompararPorPromedio);
```

44

44

Generar números aleatorios

- Utilizar la clase **`Random`** (ubicada en el namespace **`System`**)

```
Random aleatorio = new Random();
int intNumero = aleatorio.Next();
```

- El método **`Next( )`** genera un número aleatorio entre 0 y la constante **`Int32.MaxValue`** (2, 147, 483, 647)

45

45

Generar números aleatorioa

- Para generar números entre 0 y 5

```
Random aleatorio = new Random();
int intValor = aleatorio.Next(6);
```

- Para generar números entre 1 y 6

```
Random aleatorio = new Random();
int intValor = aleatorio.Next(1,7);
```

46

46

Generar cadenas aleatorias

```
string strCadena =
Guid.NewGuid().ToString().Substring(0,Longitud);
```

La variable Longitud representa el tamaño de la cadena

47

47

¿Cómo cargar datos preliminares?

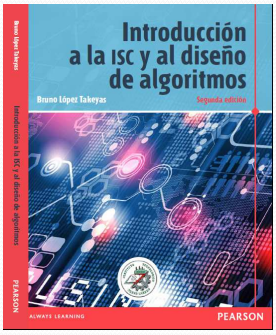
```
Estudiante miEstudiante; // Declaración del objeto
Random aleatorio = new Random(); // Declaración del número aleatorio
for (int i = 0; i < 1000; i++) // Genera 1000 objetos con datos aleatorios
{
   miEstudiante = new Estudiante(); // Crea el objeto miEstudiante

   // Genera de manera  aleatoria los datos de miEstudiante
   miEstudiante.Matricula = Guid.NewGuid().ToString().Substring(0, 9);
   miEstudiante.Nombre = Guid.NewGuid().ToString().Substring(0, 25);
   miEstudiante.Grado = aleatorio.Next(0, 14);
   miEstudiante.Grupo =
  char.Parse(Guid.NewGuid().ToString().Substring(0, 1));
   miEstudiante.Promedio = aleatorio.Next(0, 100);

   // Agrega los datos de miEstudiante al dataGridView dgEstudiantes
   dgEstudiantes.Rows.Add(miEstudiante.Matricula, miEstudiante.Nombre,
  miEstudiante.Grado.ToString(), miEstudiante.Grupo.ToString(),
  miEstudiante.Promedio.ToString("N2"));
}
```

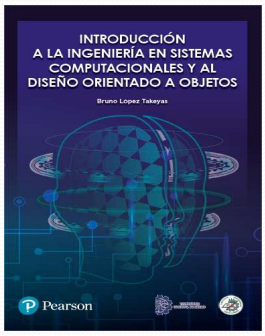
48

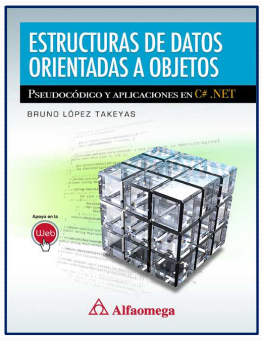
48

Otros títulos del autor

https://nlaredo.tecnm.mx/takeyas/Libro







bruno.lt@nlaredo.tecnm.mx

Bruno López Takeyas

49