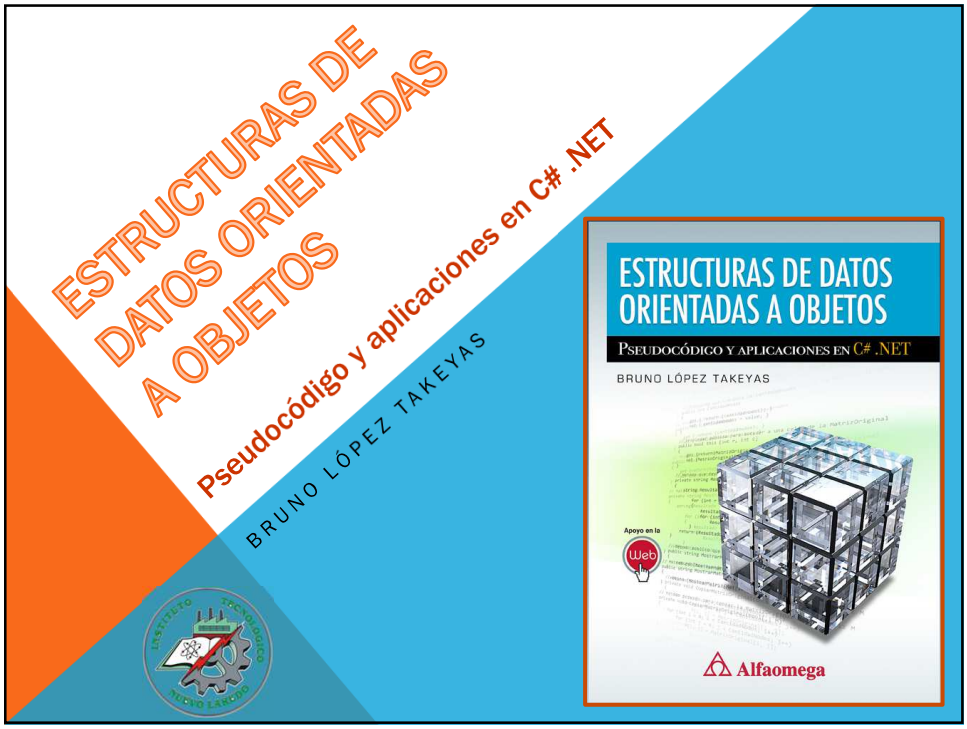
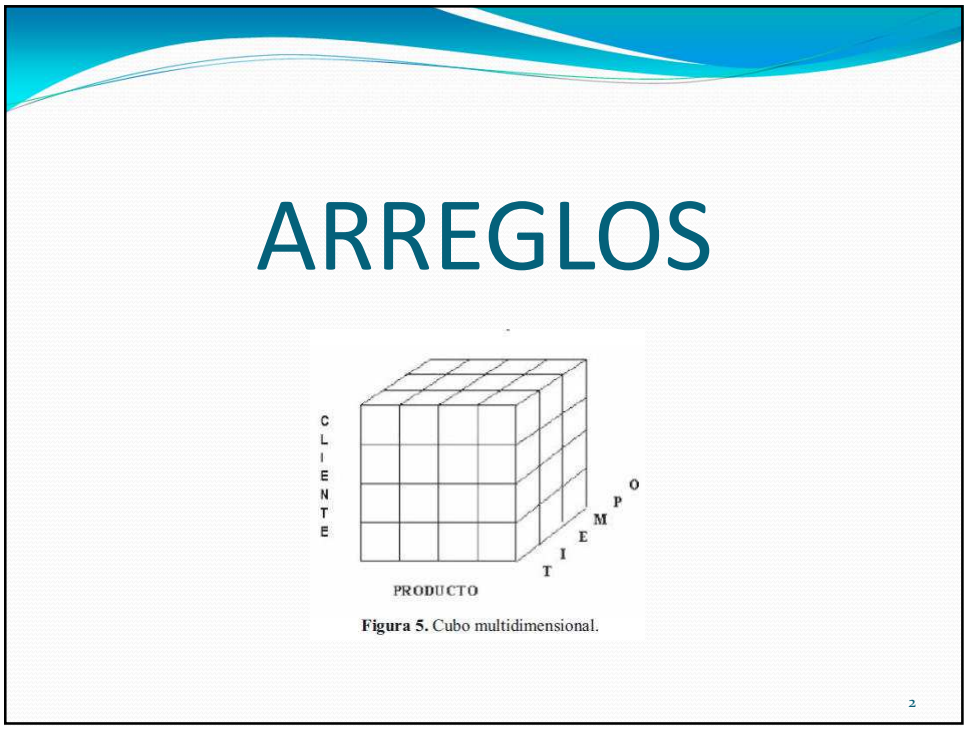

ESTRUCTURAS DE DATOS ORIENTADAS A OBJETOS
Pseudocódigo y aplicaciones en C# .NET
BRUNO LÓPEZ TAKEYAS
ESTRUCTURAS DE DATOS ORIENTADAS A OBJETOS
PSEUDOCÓDIGO Y APLICACIONES EN C# .NET
BRUNO LÓPEZ TAKEYAS
Apoyo en la Web
Alfaomega

1


ARREGLOS
CLIENTE
TIEMPO
PRODUCTO
Figura 5. Cubo multidimensional.
2

2

Preguntas detonadoras

- ¿Qué es un arreglo? ¿Para qué sirve?
- ¿Cuáles son las características que distinguen a los arreglos?
- Un arreglo, ¿puede almacenar objetos?
- Un arreglo, ¿puede ser un atributo de una clase?
- ¿Cómo se acceden a los datos de un arreglo?
- ¿Se pueden acceder a los datos de un arreglo mediante una propiedad?
- ¿Cómo se recorren las celdas de un arreglo?
- ¿Cuáles son las operaciones básicas con arreglos?

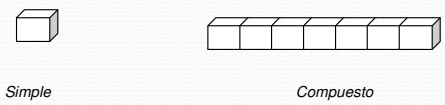
3

3

Tipos de datos

TIPOS DE DATOS

- Simples – Almacenan un valor
- Compuestos – Almacenan un conjunto de valores



Simple *Compuesto*

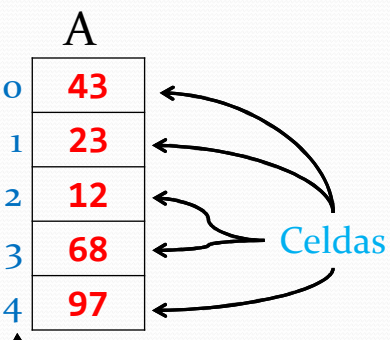
Fig. 3.1. Tipos de datos simples y compuestos.

4

4

Arreglos

Conjunto finito,
homogéneo
y estático
de datos relacionados
e indexados



Homogéneo significa que todas sus celdas son del mismo tipo de dato

Estático se refiere a que, una vez declarado, no cambia su tamaño

5

5

Dimensiones de arreglos

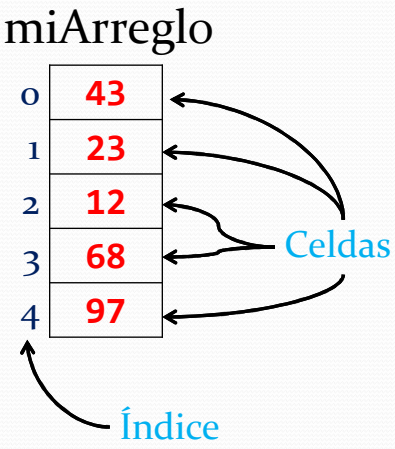
- Unidimensionales (Vectores)
  - `1 índice`
- Bidimensionales (Matrices)
  - `2 índices (renglón y columna)`
- Tridimensionales (Cubos)
  - `3 índices (renglón, columna y plano)`
- Multidimensionales

6

6

Arreglos unidimensionales

- Solamente tienen 1 índice
- También se conocen con el nombre de "vectores"



7

7

Operaciones básicas con arreglos

Operaciones básicas con arreglos

- Declaración y creación
- Inicialización
- Inserción
- Eliminación
- Búsqueda
- Ordenamiento
- Recorrido

Recuerde que **NO** se pueden agregar o eliminar celdas del arreglo; solamente su contenido

8

8

Declaración de arreglos

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

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

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

9

9

Inicialización de arreglos

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

string [ ] nombres = { "Emilio", "Pedro" };

double [ ] otroArreglo = { 3.4, -4.5, 7.0 };
```

10

10

Manejo del tamaño del arreglo

- Declarar una constante con el tamaño del arreglo

```
const int intTamano = 15;

// declara el arreglo
int [ ]  miArreglo;

// reserva memoria
miArreglo = new int[intTamano];
```

11

11

Creación de arreglos

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

```
// variable
int intTamano = 15;

// declara el arreglo
int [ ]  miArreglo;

// reserva memoria
miArreglo = new int[intTamano];
```

12

12

Capturar el tamaño del arreglo

- Se pueden crear arreglos capturando el tamaño del arreglo

```
// variable
int intTamano;

// captura el tamaño del arreglo
Console.Write("Teclee el tamaño del arreglo: ");
intTamano=int.Parse(Console.ReadLine());

// declara y crea el arreglo
int [ ]  miArreglo = new int[intTamano];
```

13

Capturar los datos de las celdas del arreglo

- Suponga que declara un arreglo que almacena datos numéricos enteros
- El usuario captura la cantidad de celdas
- Se crea el arreglo de acuerdo a la cantidad de celdas
- Se implementa un ciclo para capturar el dato de cada una de las celdas

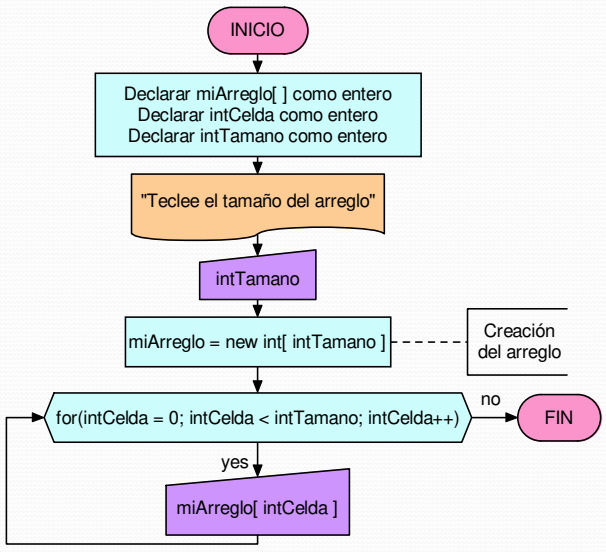
miArreglo

0	?
1	?
2	?
3	?
4	?

14

14

Diagrama de flujo



15

15

Recorrido de arreglos

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

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

for(int i=0; i < miArreglo.Length; i++)
  Console.WriteLine(miArreglo[i]);
```

16

16

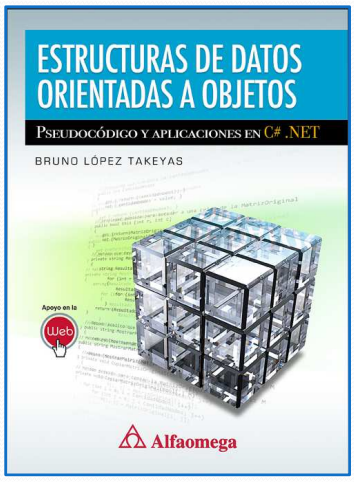
Recorrido de un arreglo con un ciclo *foreach*

```
string[] miArreglo = {"Pepe", "Paola"};

foreach(string strNombre in miArreglo)
{
    Console.WriteLine(strNombre);
}
```

17

17

LECTURA



Para consultar este tema con mayor profundidad se recomienda la lectura de:

Capítulo 3.- Arreglos

18

18

Ordenamiento de los datos de un arreglo

- Los algoritmos que ordenan datos de los arreglos se conocen como "sorteadores" o "métodos de ordenamiento"
- Existen muchos métodos de ordenamiento
- Cada método de ordenamiento tiene una estrategia para reacomodar sus datos
- No se "mueven" las celdas del arreglo, sino su contenido

19

19

Método de la burbuja

- Es el método de ordenamiento de datos más sencillo de implementar
- Su nombre se debe a que durante el ordenamiento de los datos el elemento a ordenar "burbujea" hacia arriba en el arreglo
- Existen varias versiones:
  - *Burbuja izquierda*
  - *Burbuja derecha*
  - *Burbuja con señal*

20

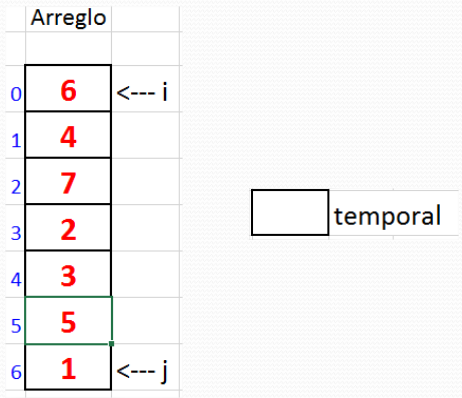
Principio del funcionamiento de la burbuja izquierda

- Recorre el arreglo de derecha a izquierda
- Compara los datos adyacentes en el arreglo
- Hace los intercambios pertinentes
- Implementa un ciclo que controla la posición que ocupará el dato del arreglo (i)
- Dentro de este ciclo hay otro ciclo que recorre el arreglo de derecha a izquierda para hacer las comparaciones (j)
- Los desplazamientos de los datos se hacen hacia la izquierda del arreglo

21

Intercambio de datos

- *Algunos métodos de ordenamiento requieren intercambiar datos*
- *Para lograrlo se apoyan en una variable auxiliar (temporal)*

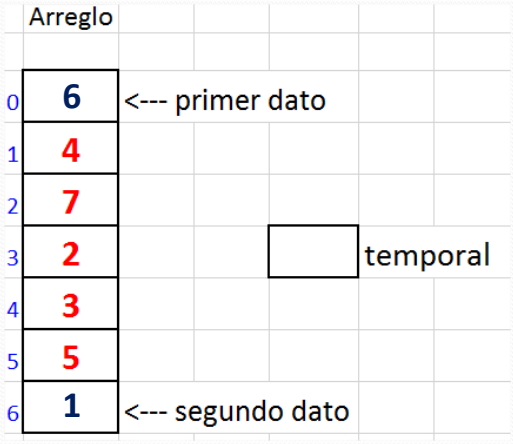


22

22

Ejemplo de intercambio de datos

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



23

23

Intercambio de datos

```
Intercambiar(Arreglo[], p, i): nulo
1.-Temporal = Arreglo[p]
2.-Arreglo[p] = Arreglo[i]
3.-Arreglo[i] = Temporal
{Fin del método}
```

24

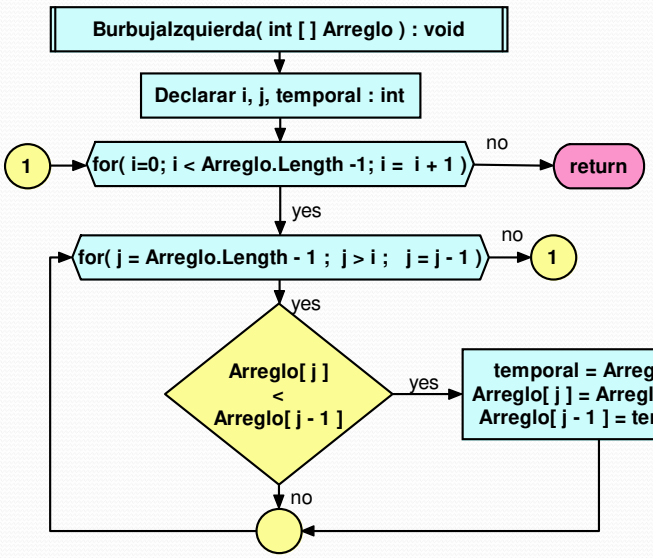
24

Pseudocódigo de la burbuja izquierda

```
BurbujaIzquierda(Arreglo[]): nulo

1.- REPETIR CON i DESDE 0 HASTA Arreglo.Length-1 CON PASO 1

    1.1. REPETIR CON j DESDE Arreglo.Length-1 HASTA i CON PASO -1
        1.1.1 SI Arreglo[j] < Arreglo[j-1] ENTONCES
                1.1.1.1 Intercambia(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
```

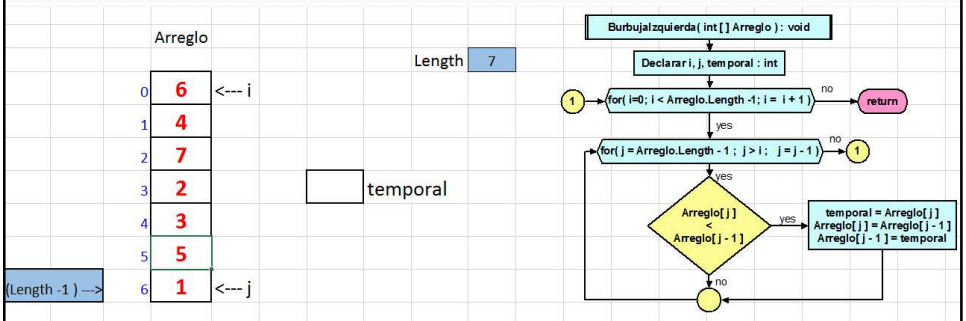
25

Diagrama de flujo del método de la burbuja izquierda que ordena números enteros



26

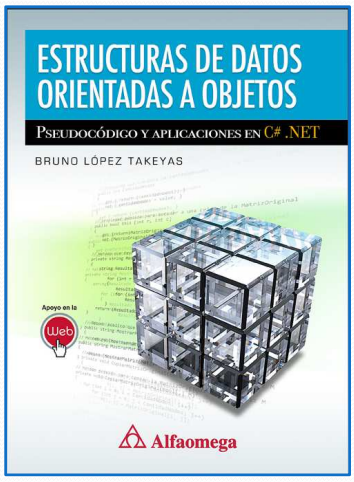
Funcionamiento del método de la burbuja



27

27

LECTURA



Para consultar este tema con mayor profundidad y revisar otros métodos de ordenamiento se recomienda la lectura de:

Capítulo 10.- Métodos de ordenamiento

28

28

¿Cómo 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)

29

29

Generar números enteros aleatorios

- Para generar números entre 0 y 5

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

- Para generar números entre 1 y 6

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

30

Generar números reales aleatorios

- Para generar números entre 0.0 y 0.1

```
Random aleatorio = new Random();
double dblNumero = aleatorio.NextDouble();
```

- Para generar números entre 0.0 y 99.99

```
Random aleatorio = new Random();
double dblNumero = aleatorio.NextDouble()*100;
```

31

Generar cadenas aleatorias

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

La variable **intLongitud** representa el tamaño de la cadena

32

32

¿Cómo llenar un arreglo con números enteros aleatorios?

```
// Declaración del arreglo
int [] miArreglo;

// Declaración del número aleatorio
Random aleatorio = new Random();

Console.Write("\nTeclee la cantidad de celdas del arreglo: ");
int intTamañoArreglo = int.Parse(Console.ReadLine());

// Creación del arreglo
miArreglo = new int[intTamañoArreglo];

// LLena el arreglo con números aleatorios
for (int intCelda = 0; intCelda < intTamañoArreglo; intCelda++)
     miArreglo[intCelda] = aleatorio.Next();
```

33

33

¿Cómo calcular el tiempo de ejecución del algoritmo?

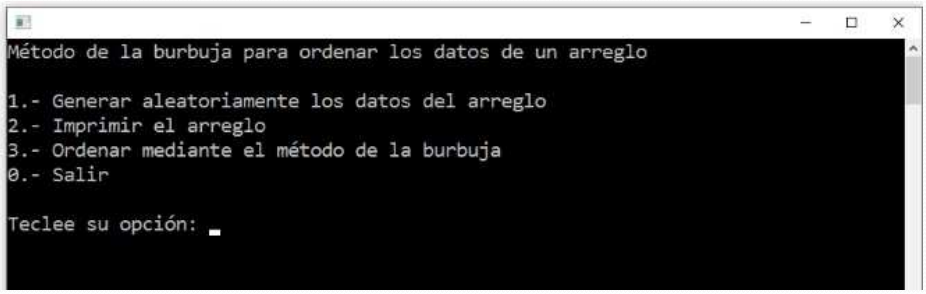
```
Console.WriteLine("Inicia cronómetro ...");
DateTime dtmHoraInicio = DateTime.Now;
.
.
.
.

DateTime dtmHoraFin = DateTime.Now;
Console.WriteLine("Finaliza cronómetro ..");

TimeSpan tiempo = dtmHoraFin - dtmHoraInicio;
Console.Write("Tiempo:"+tiempo.TotalMilliseconds+" ms.");
```

34

34

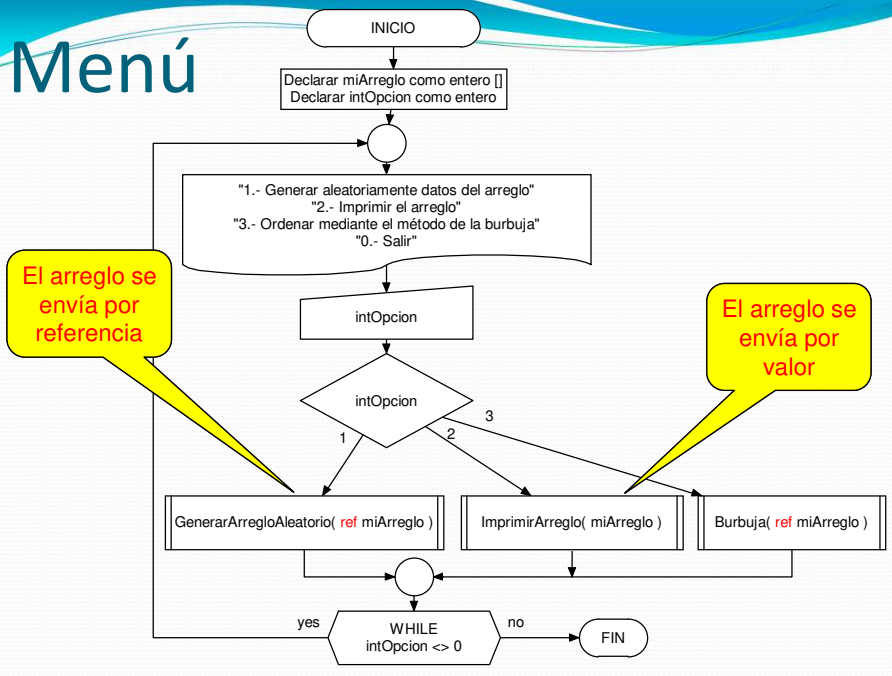
Ejercicio

- Llenar un arreglo con números aleatorios
- Enviar el arreglo por referencia a un método para ordenar sus datos
- Enviar el arreglo por valor a un método para imprimir sus datos

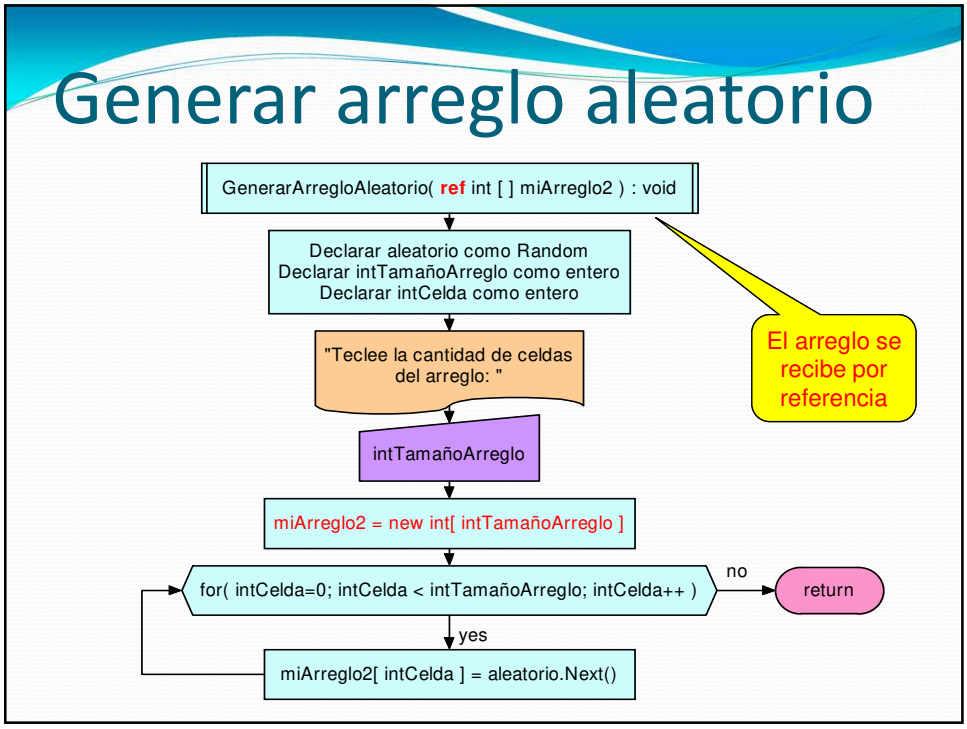


35

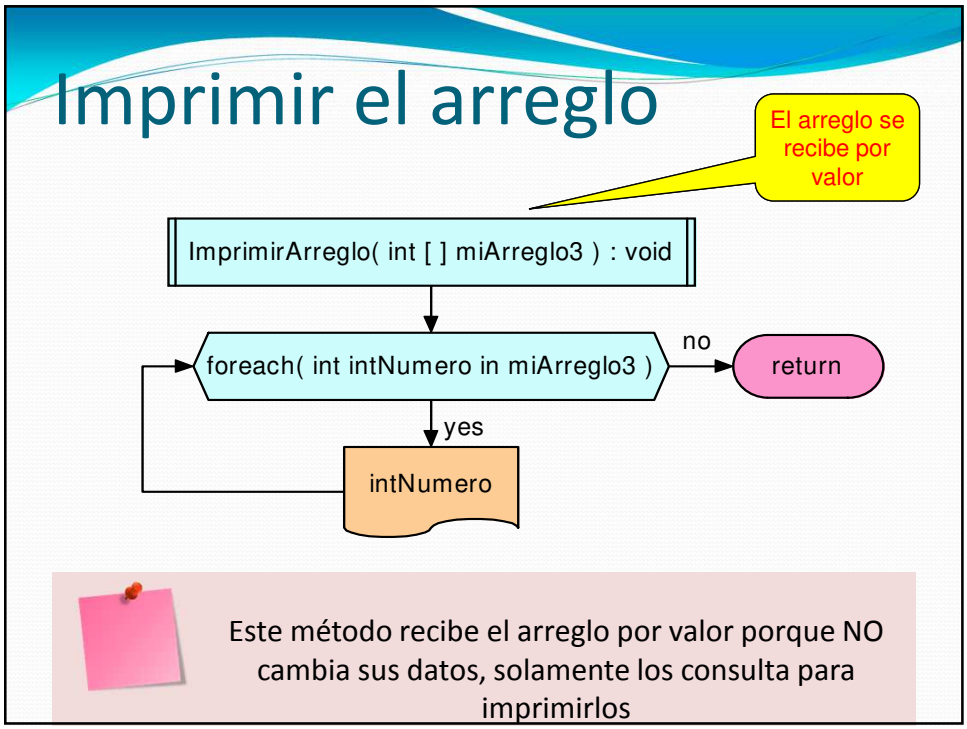
Menú



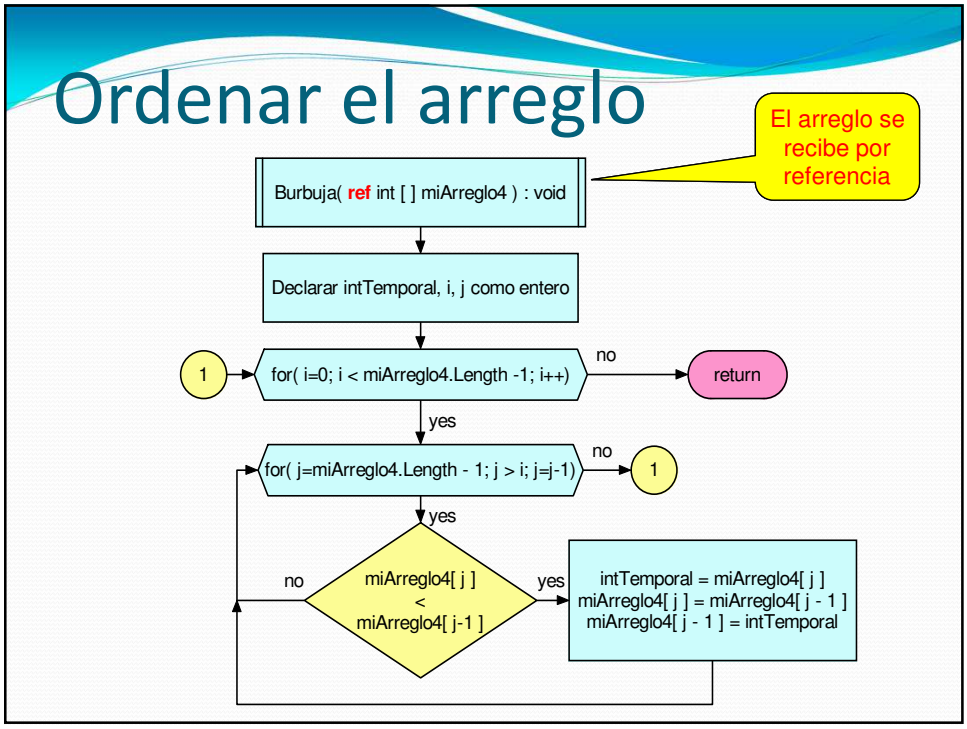
36


Generar arreglo aleatorio
GenerarArregloAleatorio(ref int [ ] miArreglo2) : void
Declarar aleatorio como Random
Declarar intTamañoArreglo como entero
Declarar intCelda como entero
"Teclee la cantidad de celdas del arreglo: "
intTamañoArreglo
miArreglo2 = new int[intTamañoArreglo]
for(intCelda=0; intCelda < intTamañoArreglo; intCelda++)
no
return
yes
miArreglo2[intCelda] = aleatorio.Next()
El arreglo se recibe por referencia

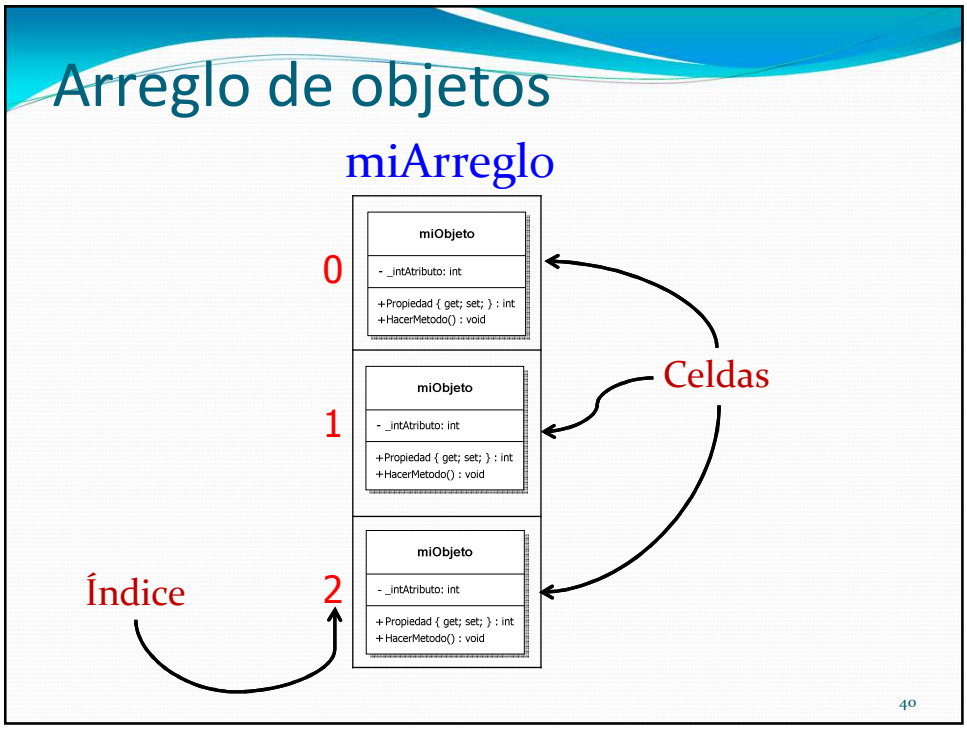
37


Imprimir el arreglo
El arreglo se recibe por valor
ImprimirArreglo(int [ ] miArreglo3) : void
foreach(int intNumero in miArreglo3)
no
return
yes
intNumero
Este método recibe el arreglo por valor porque NO cambia sus datos, solamente los consulta para imprimirlos

38


Ordenar el arreglo
El arreglo se recibe por referencia
Burbuja(ref int [ ] miArreglo4) : void
Declarar intTemporal, i, j como entero
1
for(i=0; i < miArreglo4.Length -1; i++)
no
return
yes
for(j=miArreglo4.Length - 1; j > i; j=j-1)
no
1
yes
no
miArreglo4[j] < miArreglo4[j-1]
yes
intTemporal = miArreglo4[j]
miArreglo4[j] = miArreglo4[j - 1]
miArreglo4[j - 1] = intTemporal

39


Arreglo de objetos
miArreglo
0
1
2
miObjeto
- _intAtributo: int
+Propiedad { get; set; } : int
+HacerMetodo() : void
Celdas
Índice
40

40

Declaración y creación de un arreglo de objetos

Empleado
- _intNumero: int - _strNombre: string - _dblSueldo: double
+ Numero { get; set; } : int + Nombre { get; set; } : string + Sueldo { get; set; } : double

- Suponga que desea un arreglo que almacene los empleados de una empresa
- Datos
  - *Número*
  - *Nombre*
  - *Sueldo*

41

41

Declaración y creación de un arreglo de objetos (*cont.*)

```
// Declaración del arreglo
Empleado [] miArreglo;

// Creación del arreglo
miArreglo = new Empleado[intTamaño];
```

42

42

Menú de opciones

```
ARREGLO DE OBJETOS DE EMPLEADOS

1.- Capturar el arreglo y sus objetos
2.- Imprimir los datos del arreglo

Seleccione su opción:
```

43

43

Codificación del menú de opciones

```
static void Main(string[] args)
{
   // Declaraciones de variables locales
   Empleado[] miArreglo=null;
   int intOpcion = 0;

   do  {
     Console.Clear();
     Console.WriteLine("\nARREGLO DE OBJETOS DE EMPLEADOS");
     Console.WriteLine("\n\n1.- Capturar el arreglo y sus objetos");
     Console.WriteLine("2.- Imprimir los datos del arreglo");
     Console.Write("\n\nSeleccione su opción: ");
     intOpcion = int.Parse(Console.ReadLine());
     switch(intOpcion)   {
        case 1: CapturarArreglo(ref miArreglo); break; // Por referencia
        case 2: ImprimirArreglo(miArreglo); break;     // Por valor
      }
   } while (intOpcion!=0);
}
```

44

Capturar el arreglo: Opciones…

a) Crear el objeto, capturar sus datos y luego almacenarlo en la celda del arreglo

- *Crear el arreglo especificando el tamaño*
- *Implementar un ciclo*
  - *Dentro del ciclo, crear un objeto de tipo Empleado*
  - *Capturar los datos del objeto del empleado*
  - *Almacenar el objeto en alguna celda del arreglo*
- *Iterar el ciclo*

b) Capturar los datos del objeto directamente en la celda

- *Crear el arreglo especificando el tamaño*
- *Implementar un ciclo*
  - *Dentro del ciclo, crear un objeto de tipo Empleado directamente en la celda del arreglo*
  - *Capturar los datos de un empleado directamente en la celda del arreglo*
- *Iterar el ciclo*

45

45

Captura del arreglo *(opción a)*

```
static void CapturarArreglo(ref Empleado [] nuevoArreglo)
{
   // Declaraciones locales
   int intTamaño = 0;
   Empleado nuevoEmpleado;
   Console.Clear();
   Console.WriteLine("CAPTURAR EL ARREGLO Y SUS OBJETOS\n\n");
   do  {
     Console.Write("\nTeclee el tamaño del arreglo: ");
     intTamaño = int.Parse(Console.ReadLine());
   } while (intTamaño<2);

   // Crear el arreglo según el tamaño capturado por el usuario
   nuevoArreglo = new Empleado[intTamaño];

   for(int intCelda=0; intCelda<intTamaño; intCelda++)
   {
      Console.WriteLine("\n\nCAPTURE LOS DATOS DEL EMPLEADO "+intCelda.ToString());
      Console.WriteLine();

      // Creación de un nuevo objeto para el Empleado
      nuevoEmpleado = new Empleado();

      Console.Write("Número? ");
      nuevoEmpleado.Numero = int.Parse(Console.ReadLine());

      Console.Write("Nombre? ");
      nuevoEmpleado.Nombre = Console.ReadLine();

      Console.Write("Sueldo? ");
      nuevoEmpleado.Sueldo = double.Parse(Console.ReadLine());

      // Almacenar el nuevoEmpleado en la intCelda del nuevoArreglo
      nuevoArreglo[intCelda] = nuevoEmpleado;
    }
}
```

46

46

Captura del arreglo *(opción b)*

```
static void CapturarArreglo2(ref Empleado[] nuevoArreglo)
{
   // Declaraciones locales
   int intTamaño = 0;

   Console.Clear();
   Console.WriteLine("CAPTURAR EL ARREGLO Y SUS OBJETOS\n\n");
   do  {
     Console.Write("\nTeclee el tamaño del arreglo: ");
     intTamaño = int.Parse(Console.ReadLine());
   } while (intTamaño < 2);

   // Crear el arreglo según el tamaño capturado por el usuario
   nuevoArreglo = new Empleado[intTamaño];

   for (int intCelda = 0; intCelda < intTamaño; intCelda++)  {
      Console.WriteLine("\n\nCAPTURE LOS DATOS DEL EMPLEADO " + intCelda.ToString());
      Console.WriteLine();

      // Creación de un nuevo objeto para el Empleado dentro de la celda del arreglo
      nuevoArreglo[intCelda] = new Empleado();

      Console.Write("Número? ");
      nuevoArreglo[intCelda].Numero = int.Parse(Console.ReadLine());

      Console.Write("Nombre? ");
      nuevoArreglo[intCelda].Nombre = Console.ReadLine();

       Console.Write("Sueldo? ");
       nuevoArreglo[intCelda].Sueldo = double.Parse(Console.ReadLine());
   }
}
```

47

47

Imprimir el arreglo: Opciones …

a) Recorrer automáticamente las celdas del arreglo mediante el ciclo ***foreach***

b) Recorrer las celdas del arreglo de forma "tradicional" mediante el ciclo ***for***

```
DATOS DEL ARREGLO DE OBJETOS DE EMPLEADOS

Número     Nombre                                    Sueldo
-----------------------------------------------------------
5          Bruno López Takeyas                       $55.55
4          Juan Carlos Pereyra Estrada               $44.56

Oprima cualquier tecla para continuar ...
```

48

48

Imprimir el arreglo *(opción a)*

```
static void ImprimirArreglo(Empleado [] unArreglo)
{
   Console.Clear();
   Console.WriteLine("DATOS DEL ARREGLO DE OBJETOS DE EMPLEADOS\n\n");
   Console.WriteLine("Número    Nombre                              Sueldo");
   Console.WriteLine("-----------------------------------------------------");

   foreach(Empleado unEmpleado in unArreglo)
      Console.WriteLine(unEmpleado.Numero.ToString().PadRight(6,' ')+
                  "    "+unEmpleado.Nombre.PadRight(30, ' ')+
                  "    "+unEmpleado.Sueldo.ToString("C"));

   Console.Write("\n\nOprima cualquier tecla para continuar ...");
   Console.ReadKey();
}
```

49

49

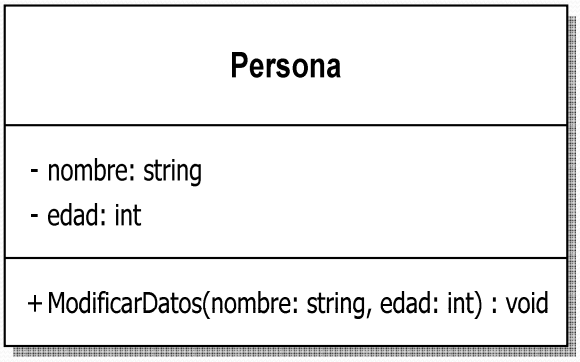
Imprimir el arreglo *(opción b)*

```
static void ImprimirArreglo2(Empleado[] unArreglo)
{
   Console.Clear();
   Console.WriteLine("DATOS DEL ARREGLO DE OBJETOS DE EMPLEADOS\n\n");
   Console.WriteLine("Número    Nombre                              Sueldo");
   Console.WriteLine("-----------------------------------------------------
");

   for (int i = 0; i < unArreglo.Length; i++)
      Console.WriteLine(unArreglo[i].Numero.ToString().PadRight(6, ' ') +
                "    " + unArreglo[i].Nombre.PadRight(30, ' ') +
                "    " + unArreglo[i].Sueldo.ToString("C"));

   Console.Write("\n\nOprima cualquier tecla para continuar ...");
   Console.ReadKey();
}
```

50

50

Ejemplo de uso de la referencia *this*



El uso de la palabra ***this*** para referirse a los miembros internos de una clase es opcional, pero es necesaria cuando un parámetro y un atributo comparten el mismo nombre

51

51

Codificación de la referencia *this*

```
class Persona
    {
        // Declaración de los atributos privados
        private string nombre;
        private int edad;

        // Mutator
        public void ModificarDatos(string nombre, int edad)
        {
            this.nombre = nombre;
            this.edad = edad;
        }
    }
```

52

La referencia *this*

- Para hacer referencia (explícita) a un elemento que se encuentra dentro de la misma clase (ésta) se utiliza "`this`".

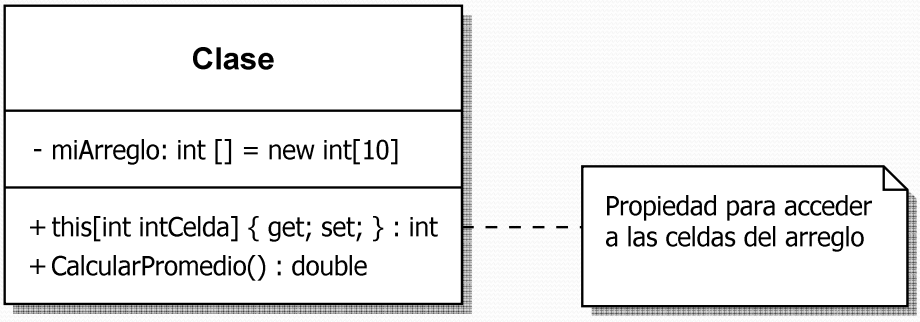
```
class Articulo
{   private double precio = 0;
    public void PonerPrecio ( double precio )
    {
        this.precio = precio;
    }
    public double ConsultarPrecio()
    {
        return this.precio;
    }
}
```

Muy útil cuando existen distintos elementos con el mismo nombre, pero con distinto significado dentro de un ámbito determinado.

53

53

Diseño de una clase con un arreglo como atributo privado



54

54

Codificación de la clase

```
class Clase
{
   // Atributo privado
   private int [] miArreglo = new int[10];

   // Propiedad (referencia this)
   public int this[int intCelda]
   {
       get { return miArreglo[ intCelda ]; }
       set { miArreglo[ intCelda ] = value; }
   }

   // Método
   public double CalcularPromedio()
   {
      . . .
   }
}
```

55

Propiedad para acceder al valor de una celda de un arreglo de una clase

```
// Declaración y creación del objeto
Clase miObjeto = new Clase();

for (int intCelda = 0; intCelda < 10; intCelda++)
{
    Console.Write("Teclee el dato {0}: ", intCelda + 1);
    miObjeto[intCelda] = int.Parse(Console.ReadLine());
}
```

56

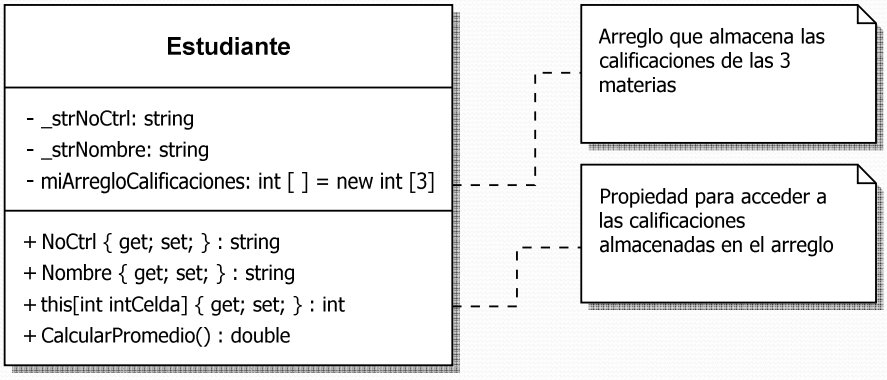
56

Ejercicio resuelto

- Un profesor requiere una aplicación para administrar los datos de sus estudiantes:
  - *Número de control (string)*
  - *Nombre (string)*
  - *Calificaciones de sus 3 materias (int)*
- Para ello, requiere un arreglo donde cada celda almacene un objeto con los datos de un estudiante

57

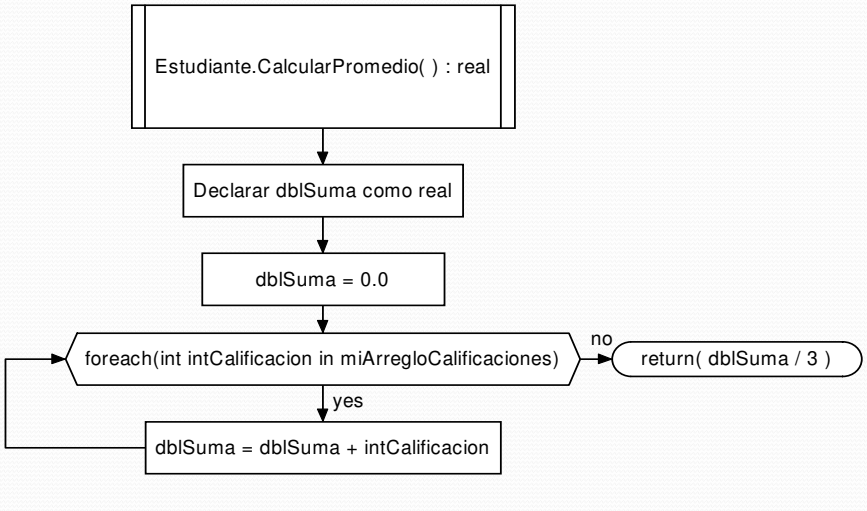
Diseño de la clase Estudiante



58

58

Diagrama de flujo del método para calcular el promedio de un estudiante

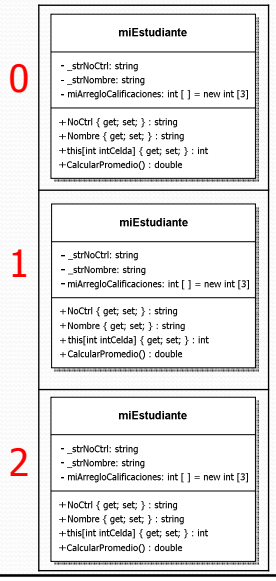


59

59

Arreglo de objetos de tipo Estudiante

miArregloEstudiantes



60

60

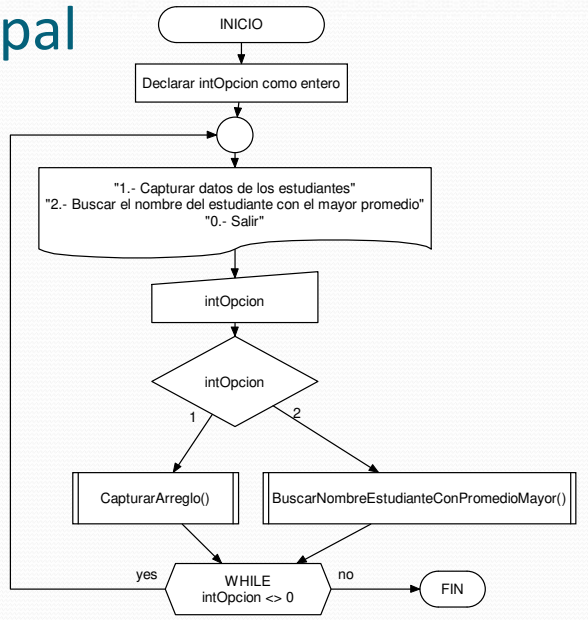
Operación de la aplicación

- Diseñe un menú con las opciones

```
MENÚ DE OPCIONES
1.- Capturar los datos de los estudiantes
2.- Nombre del estudiante con el mayor promedio
0.- Salir

Teclee su opción:
```

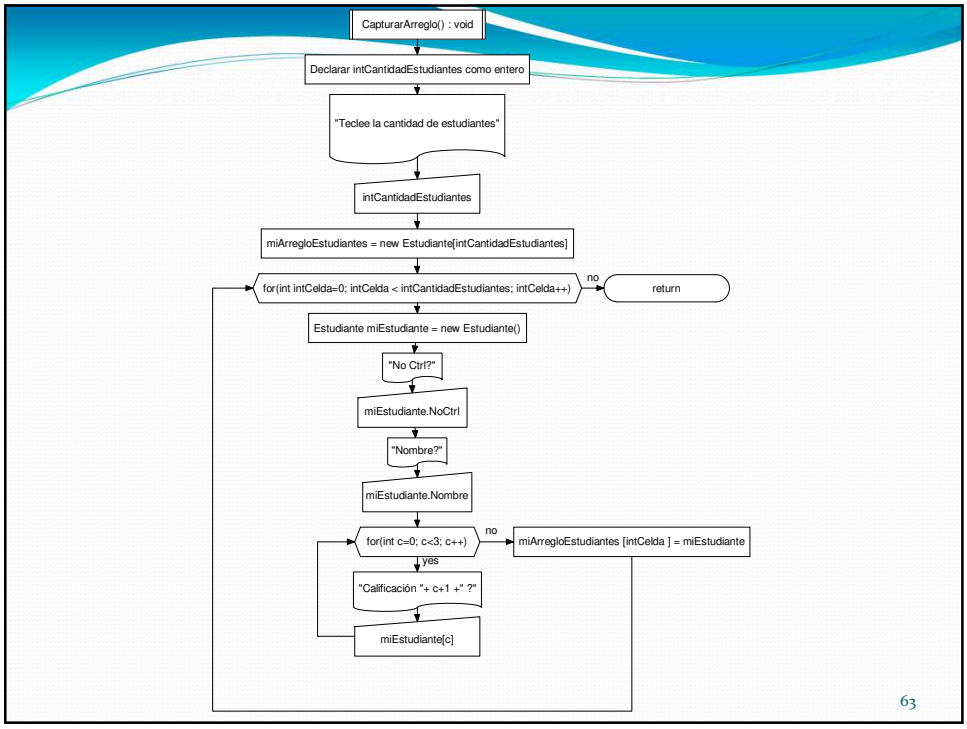
61

Diagrama de flujo del método principal



62

62


CapturarArreglo() : void
Declarar intCantidadEstudiantes como entero
"Teclee la cantidad de estudiantes"
intCantidadEstudiantes
miArregloEstudiantes = new Estudiante[intCantidadEstudiantes]
for(int intCelda=0; intCelda < intCantidadEstudiantes; intCelda++)
no
return
Estudiante miEstudiante = new Estudiante()
"No Ctrl?"
miEstudiante.NoCtrl
"Nombre?"
miEstudiante.Nombre
for(int c=0; c<3; c++)
no
miArregloEstudiantes [intCelda] = miEstudiante
yes
"Calificación "+ c+1 +" ?"
miEstudiante[c]
63

63

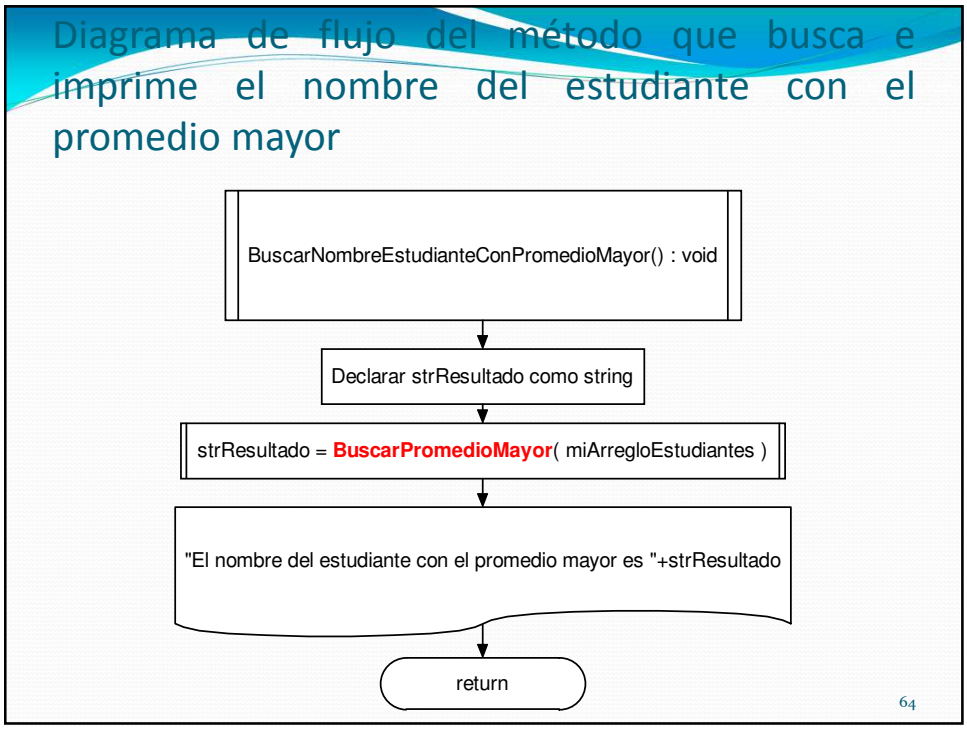
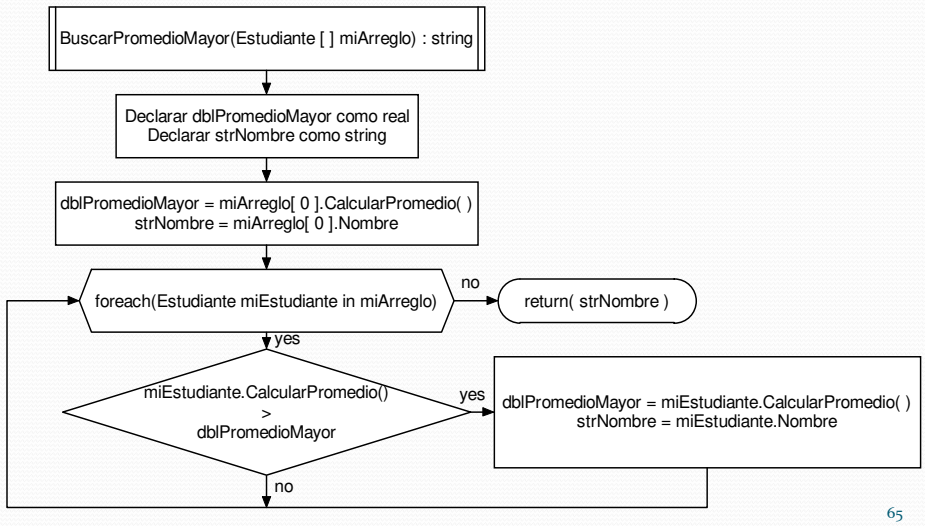


Diagrama de flujo del método que busca e imprime el nombre del estudiante con el promedio mayor
BuscarNombreEstudianteConPromedioMayor() : void
Declarar strResultado como string
strResultado = BuscarPromedioMayor(miArregloEstudiantes)
"El nombre del estudiante con el promedio mayor es "+strResultado
return
64

64

Diagrama de flujo del método que devuelve el nombre del estudiante con el promedio mayor



65

65

Arreglos bidimensionales

- Tienen 2 índices (renglón, columna)
- También se conocen con el nombre de "matrices"

miArreglo	Columna 0	Columna 1
Renglón 0	miArreglo[0,0]	miArreglo[0,1]
Renglón 1	miArreglo[1,0]	miArreglo[1,1]
Renglón 2	miArreglo[2,0]	miArreglo[2,1]

66

66

Declaración de matrices

- No sólo basta con declararlo, sino también debe crearse con el operador **new**
- Se declara primero la cantidad de renglones seguido de la cantidad de columnas

```
int [ , ]  matriz;   // declara la matriz
matriz = new int[2,3];  // reserva memoria

double [ , ] matriz2 = new double[3,4];
```

67

67

Inicialización de matrices

- Se declara e inicializa una matriz de

  3 X 2

  3 renglones

  2 columnas

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

68

68

Recorrido de matrices

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

for(int r=0; r< matriz.GetLength(0); r++)
{
   Console.Write("\n");
     for(int c=0; c< matriz[r].GetLength(1); c++)
        Console.Write("\t" + matriz[r,c]);
}
```

69

69

Recorrido de una matriz mediante el ciclo *foreach*

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

foreach(int numero in matriz)
{
   Console.WriteLine("\t" + numero);
}
// Imprime 1 2 3 4 5 6
```

70

70

Ejemplo: Menú

```
static void Main(string[] args)
{
   // Declaración de una variable local para la matriz
   int[,] miMatriz = null;

   int intOpcion = 0;
   do {
     Console.Clear();
     Console.WriteLine("M A T R I C E S\n");
     Console.WriteLine("1.- Generar datos aleatorios de la matriz");
     Console.WriteLine("2.- Imprimir matriz");
     Console.WriteLine("0.- Salir");
     Console.Write("\nTeclee su opción: ");
     intOpcion = int.Parse(Console.ReadLine());

     switch(intOpcion) {
       // Se envía por referencia
       case 1: GenerarDatosMatriz(ref miMatriz); break;
       // Se envía por valor
       case 2: ImprimirMatriz(miMatriz); break;
      }
    } while (intOpcion!=0);
}
```

71

Método para llenar la matriz

```
// Método que recibe la matriz por referencia para llenarla con datos aleatorios
       static void GenerarDatosMatriz(ref int [,] matriz)
       {
           int intRenglones = 0, intColumnas = 0;

           Console.Clear();
           Console.WriteLine("Capture los datos de la matriz");
           Console.Write("\n\n¿Cantidad de renglones? ");
           intRenglones = int.Parse(Console.ReadLine());

           Console.Write("\n¿Cantidad de columnas? ");
           intColumnas = int.Parse(Console.ReadLine());

           // Creación de la matriz
           matriz = new int[intRenglones, intColumnas];

           Random aletorio = new Random();

           // Generar datos aleatorios para llenar la matriz
           for (int r = 0; r < intRenglones; r++)
               for (int c = 0; c < intColumnas; c++)
                   matriz[r, c] = aletorio.Next(100);

           Console.WriteLine("\n\nSe ha generado la matriz con datos aleatorios");
           Console.WriteLine("\nOprima cualquier tecla para continuar...");
           Console.ReadKey();
       }
```

72

Método para imprimir la matriz

```
// Método que recibe la matriz por valor para imprimir sus
datos
static void ImprimirMatriz(int [,] matriz)
{
   Console.Clear();
   Console.WriteLine("Datos de la matriz");

   for(int r=0; r<matriz.GetLength(0); r++)
   {
      Console.Write("\n");
      for (int c = 0; c < matriz.GetLength(1); c++)
           Console.Write(matriz[r, c].ToString() + "\t"); ;
   }
}
```

73

73

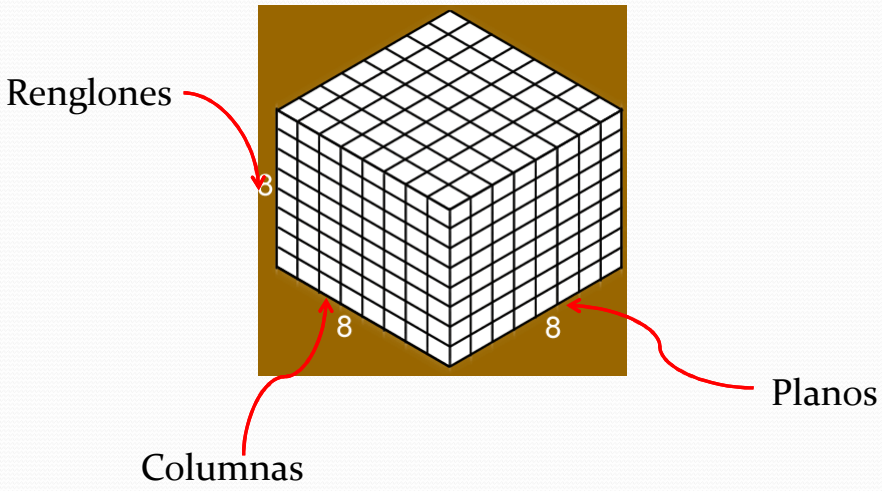
Ejemplo de salida

```
Datos de la matriz

15      74      8       15
57      97      2       99
30      2       60      48

Oprima cualquier tecla para continuar...
```

74

74

Arreglos trimensionales (cubos)



75

75

Declaración de cubos

- No sólo basta con declararlo, sino también debe crearse con el operador **new**
- Se declara primero la cantidad de renglones seguido de la cantidad de columnas y luego los planos

```
int [ , , ]  miCubo;   // declara el cubo
miCubo = new int[2,3,4];  // reserva memoria

double [ , , ] miCubo2 = new double[3,4,5];
```

76

76

Manejo de cubos

```
int[, ,] miCubo = new int[2, 3, 4];

for(int r=0; r < miCubo.GetLength(0); r++)
 for(int c=0; c < miCubo.GetLength(1); c++)
  for (int p=0; p < miCubo.GetLength(2); p++)
  {
      Console.Write("\nCubo[{0},{1},{2}] ---> ", r, c, p);
      miCubo[r, c, p] = int.Parse(Console.ReadLine());
  }
```

77

77

Recorrido de un cubo mediante el ciclo *foreach*

```
int [ , , ] miCubo = new int [2,3,4];

foreach(int numero in miCubo)
{
   Console.WriteLine("\t" + numero);
}
```

78

78

Recorrido de un cubo mediante ciclos *for*

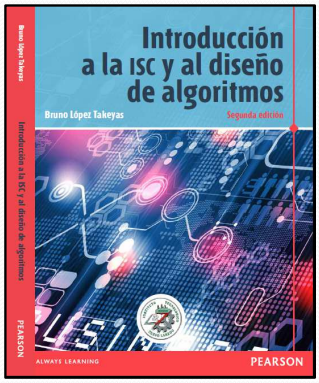
```
int [ , , ] miCubo = new int [2,3,4];

for(int r=0; r < miCubo.GetLength(0); r++)
  for(int c=0; c < miCubo.GetLength(1); c++)
    for(int p=0; p < miCubo.GetLength(2); p++)
       Console.WriteLine( miCubo[r, c, p] );
```

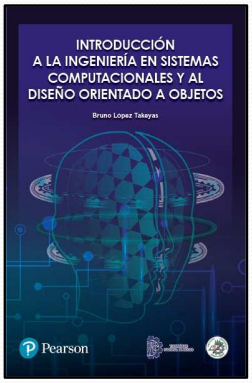
79

79

Otros libros del autor

http://www.itnuevolaredo.edu.mx/Takeyas/Libro





bruno.lt@nlaredo.tecnm.mx

Bruno López Takeyas

80