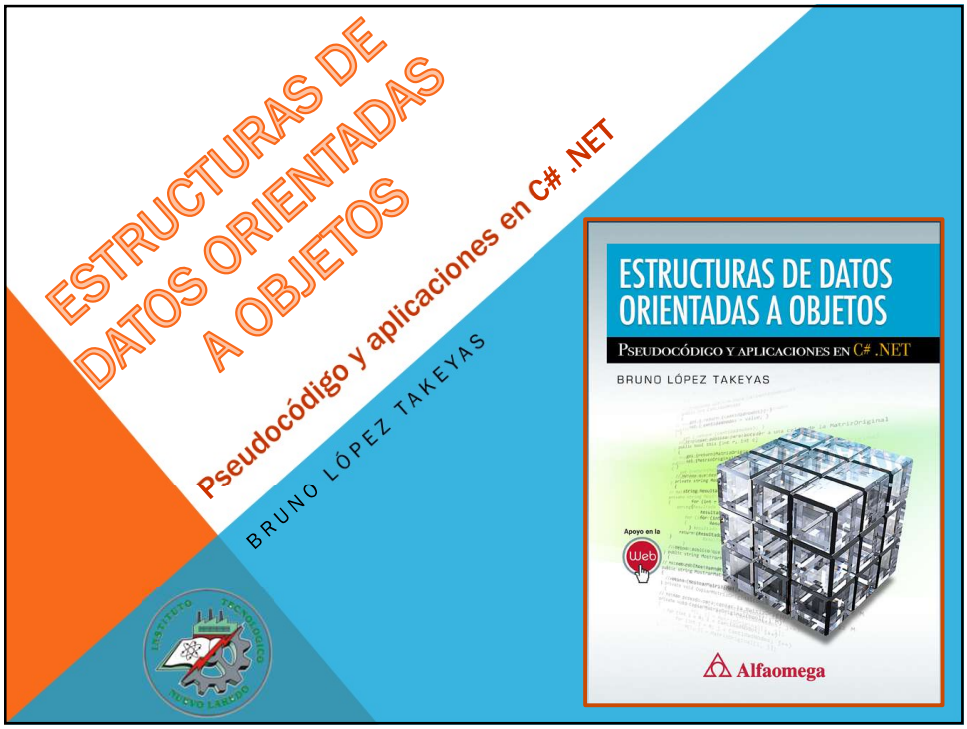
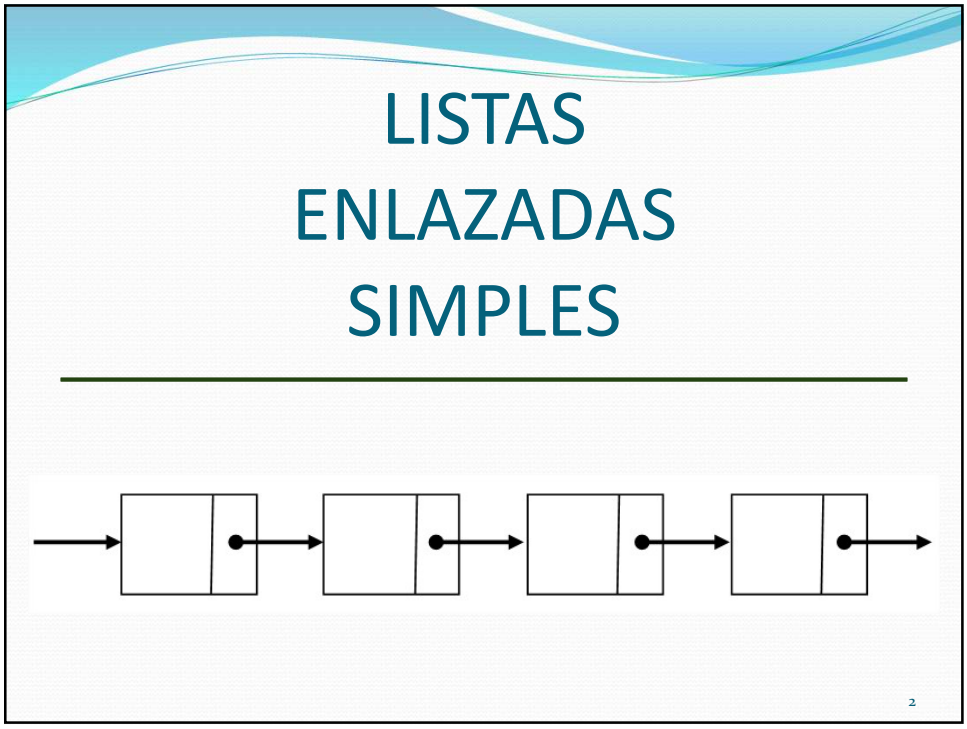

ESTRUCTURAS DE DATOS ORIENTADAS A OBJETOS
Pseudocódigo y aplicaciones en C# .NET
BRUNO LÓPEZ TAKEYAS
ESTRUCTURAS DE DATOS ORIENTADAS A OBJETOS
PSEUDOCÓDIGO Y APLICACIONES EN C# .NET
BRUNO LÓPEZ TAKEYAS
Alfaomega


LISTAS ENLAZADAS SIMPLES
2

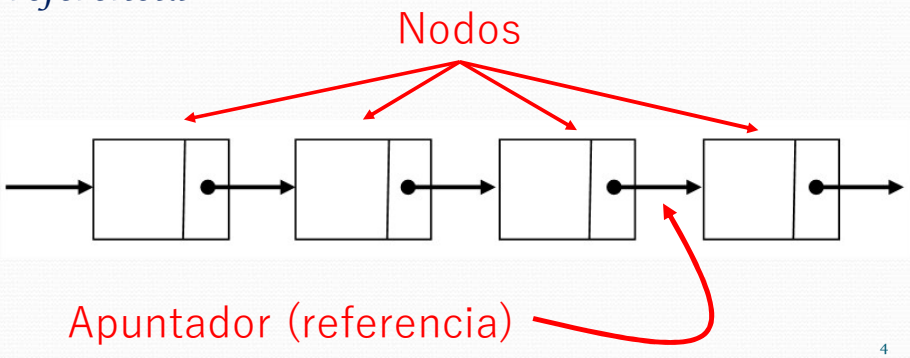
Preguntas detonadoras

- ¿Qué es una lista enlazada?
- ¿Cómo se clasifican las listas enlazadas?
- ¿Qué características distinguen a una lista enlazada simple?
- ¿Cómo se representa gráficamente una lista enlazada simple?
- ¿Qué operaciones se pueden realizar en una lista enlazada simple?
- ¿Por qué es una estructura dinámica?
- ¿Cómo se diseña un modelo orientado a objetos con una lista simple?

3

Lista enlazada simple

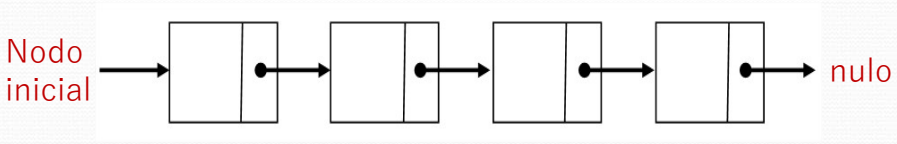
Es una estructura de datos lineal y dinámica que se compone de un conjunto de nodos en secuencia enlazados mediante un apuntador o referencia



4

Representación gráfica de una lista simple

- *La lista simple tiene un apuntador inicial*
- *El último nodo de la lista apunta a nulo*



5

Tipos de listas simples

Listas simples

- *Con datos ordenados*
- *Con datos desordenados*

6

Almacenamiento de los datos en una lista simple

- Ordenados:
  - *Se recorren **lógicamente** los nodos de la lista simple*
  - *Se ubica la posición definitiva de un nuevo nodo*
  - *Se mantiene el orden **lógico** de los datos*
- Desordenados
  - *El nuevo dato se agrega al final de la lista simple*

7

Ejemplos de listas en la vida cotidiana

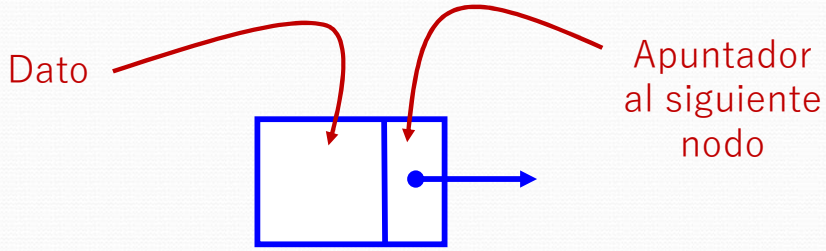
- *Lista de estudiantes ordenada alfabéticamente (independientemente del orden en el que se inscribieron)*
- *Se pueden utilizar listas como base para implementar otras estructuras de datos como pilas, colas, árboles, grafos, etc.*

8

Arquitectura de un nodo

Cada nodo tiene 2 secciones:

1. *Dato (puede ser simple o compuesto)*
2. *Apuntador o referencia que enlaza al siguiente nodo en secuencia lógica*



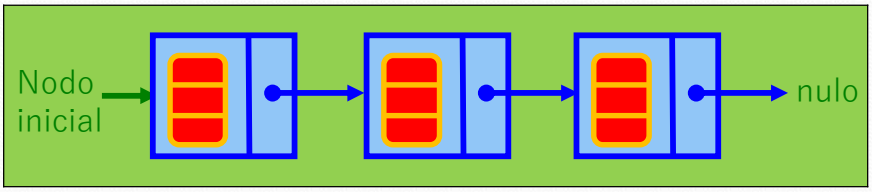
9

Diseño de una lista simple orientada a objetos

- Se identifican 3 tipos de objetos en la lista simple:
  1) *Los objetos con los datos que se desean almacenar y ordenar*
  2) *Los objetos de los nodos*
  3) *El objeto de la lista simple ordenada*

10

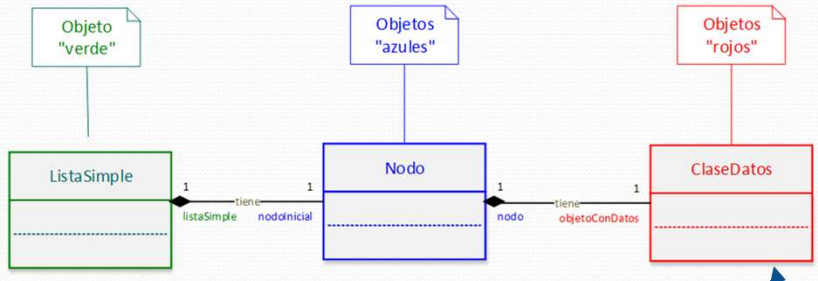
Esquema de los objetos



La **lista simple (objeto verde)** dentro tiene una secuencia lógica de **nodos (objetos azules)** enlazados por apuntadores y cada uno de ellos a su vez tiene dentro un **objeto con los datos** que se desean almacenar y ordenar **(objetos rojos)**

11

Diseño genérico y didáctico de la lista simple



- *Esta clase puede ser...*
- `Empleado`
- `Escuela`
- `Médico`
- *etc. (según lo que se desee almacenar en la lista simple)*

12

Diseño orientado a objetos de una lista simple ordenada

- Se diseña una lista simple con objetos creados por medio de varios tipos de clases:
  - *Clase "roja".- Sirve para crear objetos con los datos que se desean almacenar y ordenar en la lista*
  - *Clase "azul".- Clase para crear los nodos*
  - *Clase "verde".- Clase para crear el objeto de la lista simple*

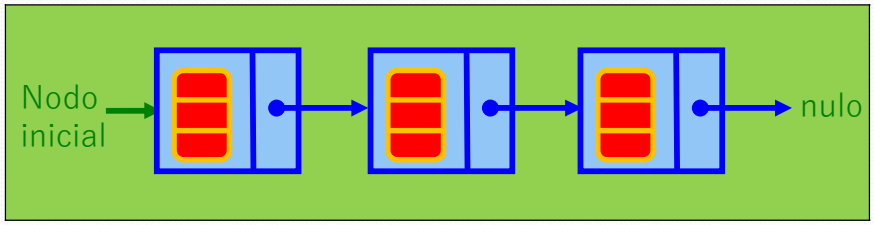
13

Notación de colores

- Los estudiantes deben poner especial atención a los colores usados en las figuras explicativas del resto del curso.
- Los objetos estarán identificados por colores
  - *Objetos "rojos".- Contienen los datos que se desean almacenar y ordenar en la lista*
  - *Objetos "azules".- Representan los nodos de la lista simple*
  - *Objeto "verde".- Es la lista simple*

14

Diseño de clases



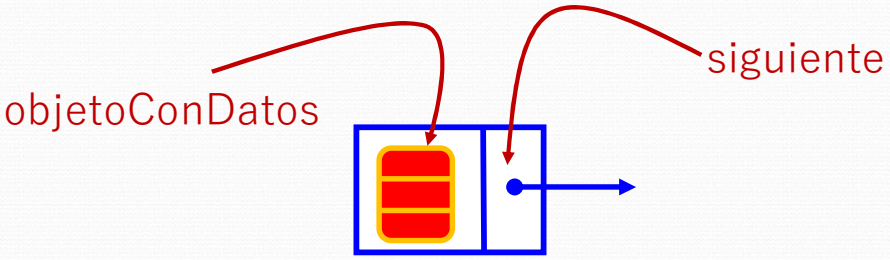
- **Clase "verde"**.- *Define la **composición** entre la lista simple y el nodo inicial. También tiene los métodos y propiedades para administrar la lista.*
- **Clase "azul"**.- *Define los componentes de los nodos.*
- **Clase "roja"**.- *Definiciones de los datos que se desean almacenar y ordenar en la lista simple.*

15

Declaración del nodo

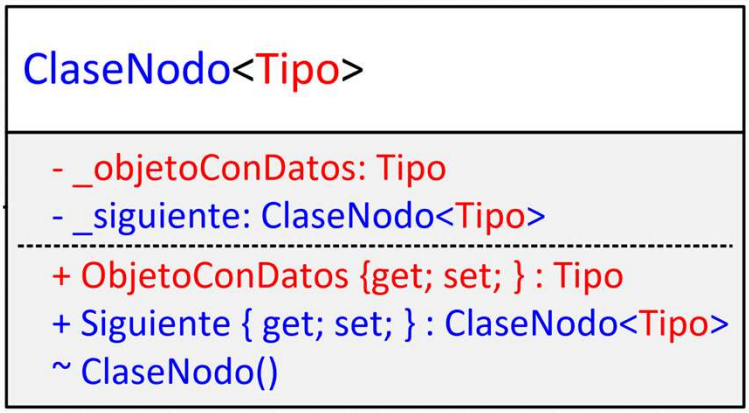
Cada nodo tiene 2 atributos (con sus propiedades):

1. ***_objetoConDatos***.- *El nodo recibirá un objeto con los datos que se desean almacenar en la lista simple.*
2. ***_siguiente***.- *Apuntador que enlaza al siguiente nodo lógico en la lista simple*



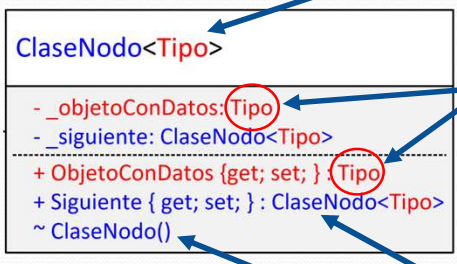
16

Diseño de la clase de los nodos



17

Diseño de la `ClaseNodo`

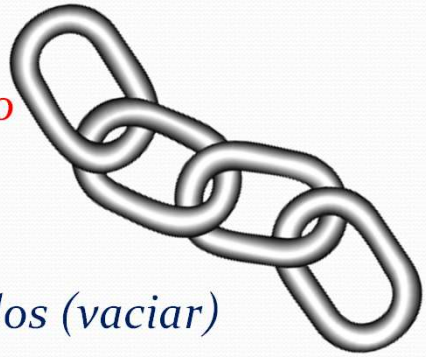


- *Clase parametrizada para recibir cualquier tipo de objeto "rojo"*
- *El parámetro `<Tipo>` define el tipo de objeto "rojo" que estará dentro de cada nodo "azul"*
- *El apuntador `Siguiente` NO almacena otro nodo "azul" sino apunta hacia otro nodo de su mismo tipo*
- *Al eliminar un nodo "azul", su destructor elimina el objeto "rojo" que contiene*

18

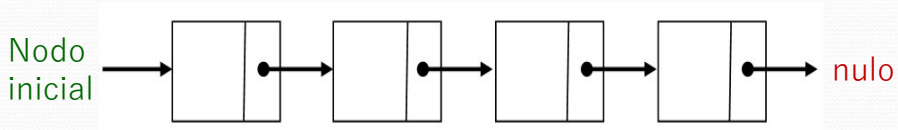
Operaciones en una lista simple

- *Creación de la lista*
- *Inserción de un dato*
- *Eliminación de un dato*
- *Recorrido*
- *Búsqueda*
- *Eliminar todos los nodos (vaciar)*



19

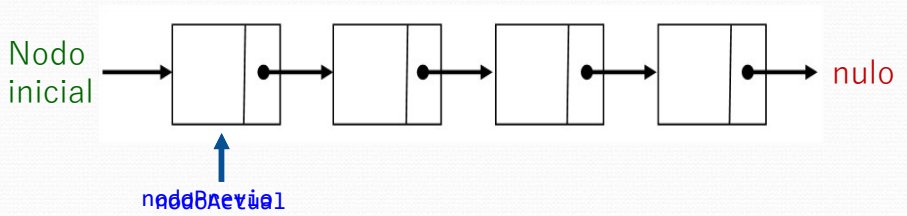
Recorrido de los nodos en una lista simple



- *Se verifica que la lista no esté vacía*
- *El recorrido empieza en el* `NodoInicial`
- *Se avanza al próximo nodo a través del apuntador* `Siguiente`
- *En algunos casos es necesario guardar en una variable el nodo previo al cambiar al siguiente nodo*
- *El recorrido termina al llegar al nodo que apunta a nulo*

20

Ejemplo de recorrido

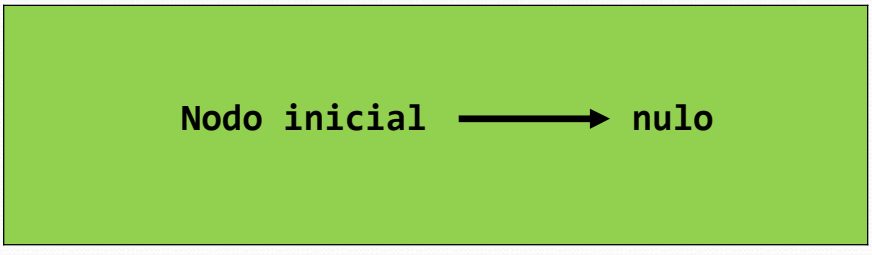


Termina el recorrido

21

Creación de una lista simple ordenada

Cuando se crea una lista simple el nodo inicial apunta a nulo



22

Situaciones críticas

- *Son las situaciones que se pueden presentar al realizar operaciones con estructuras de datos*
- *El programador debe prever para diseñar algoritmos eficientes*

23

Inserción de datos en una lista simple ordenada

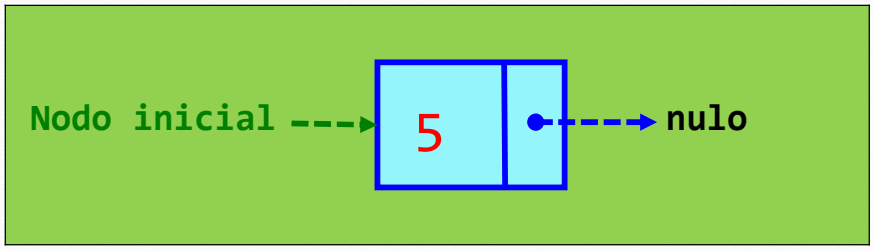
- Situaciones críticas:
  - *Alta a lista vacía.- Se inserta el primer dato*
  - *Alta al principio.- Se inserta el dato menor*
  - *Alta intermedia.- Se inserta un dato entre otros*
  - *Alta al final.- Se inserta el dato mayor*
  - ***NO** permitir datos duplicados*

24

Alta a una lista simple ordenada vacía

Cuando se detecta la lista simple vacía:

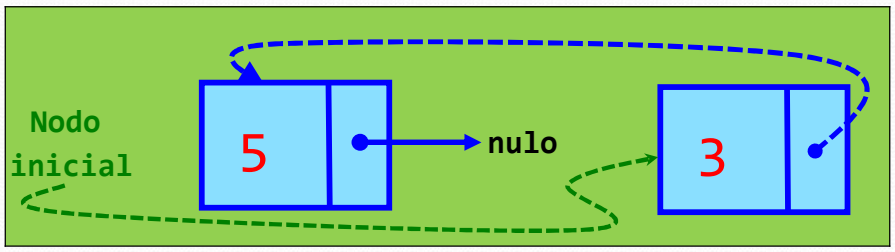
1) *Crear el nodo nuevo "azul"*
2) *Insertar el dato "rojo"*
3) *El nodo nuevo apunta a nulo*
4) *El nodo inicial apunta al nodo nuevo*



Alta al principio en una lista simple ordenada

Cuando se inserta el dato menor:

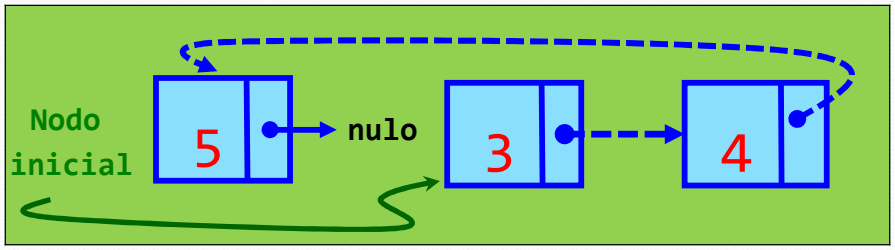
1) *Crear el nodo nuevo "azul"*
2) *Insertar el dato "rojo"*
3) *El nodo nuevo apunta al nodo inicial*
4) *El nodo inicial apunta al nodo nuevo*



Alta intermedia en una lista simple ordenada

Cuando se inserta un dato intermedio:

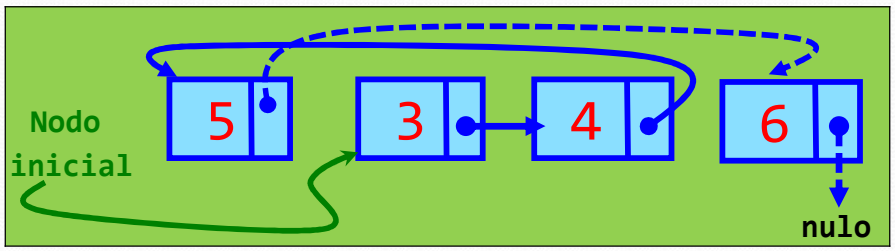
1) *Crear el nodo nuevo "azul"*
2) *Insertar el dato "rojo"*
3) *El nodo nuevo apunta al que apuntaba el nodo previo*
4) *El nodo previo apunta al nodo nuevo*



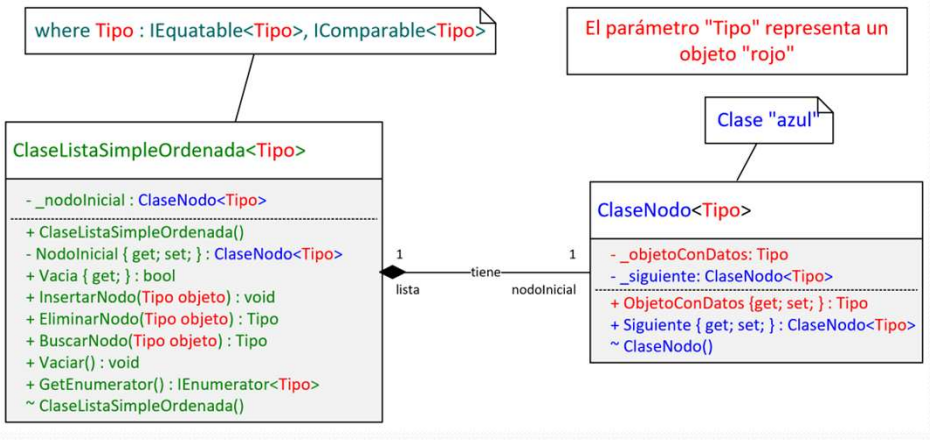
Alta al final en una lista simple ordenada

Cuando se inserta un dato al final:

1) *Crear el nodo nuevo "azul"*
2) *Insertar el dato "rojo"*
3) *El nodo previo apunta al nodo nuevo*
4) *El nodo nuevo apunta a nulo*



Diseño de la clase de la lista simple



29

Composición Lista-Nodo

- *¿Por qué es una composición **1..1** si la lista tiene muchos nodos dentro de ella?*
  - *Porque la cardinalidad de una composición se define por la cantidad de atributos de tipo "parte" contenidos en la clase del "todo" (regla 1 de la composición)*

30

ClaseListaSimpleOrdenada<Tipo>

- *Clase parametrizada para prepararla para que pueda recibir cualquier `Tipo` de objeto "rojo"*
- *Tiene una restricción de tipos para "obligar" a que la clase "roja" implemente `IEquatable` e `IComparable`.*
  - *Se requiere el método `Equals()` para buscar un objeto "rojo" almacenado en la lista*
  - *Se requiere el método `CompareTo()` para ordenar los objetos "rojos"*

31

Componentes de la clase

- `_nodoInicial`.- *Atributo privado que apunta al primer nodo de la lista simple*
- `ClaseListaSimpleOrdenada()`.- *Es el constructor que inicializa vacía la lista.*
- `Vacia`.- *Propiedad pública booleana de solo lectura para detectar si la lista está vacía (devuelve `true` si la lista está vacía).*

32

Componentes de la clase *(cont.)*

- `NodoInicial`.- *Propiedad privada que apunta al primer nodo de la lista simple*
- `AgregarNodo(Tipo objeto):void` .- *Método público que recibe como parámetro el objeto "rojo" que se desea almacenar en la lista.*
- `EliminarNodo(Tipo objeto):Tipo` .- *Método público que recibe como parámetro el objeto "rojo" que se desea borrar de la lista.* ***Devuelve*** *el objeto "rojo" eliminado.*

33

Componentes de la clase *(cont.)*

- `BuscarNodo(Tipo objeto):Tipo` .- *Método público que recibe como parámetro el objeto "rojo" que se desea consultar en la lista.* ***Devuelve*** *el objeto "rojo" localizado.*
- `Vaciar():void` .- *Método público que recorre la lista para eliminar todos los nodos "azules" con sus respectivos objetos "rojos"*

34

Componentes de la clase *(cont.)*

- `GetEnumerator():IEnumerator<Tipo>`.- *Iterador que recorre cada nodo "azul" de la lista para consultar su objeto "rojo"*

- `~ClaseListaSimpleOrdenada()`.- *Destructor que invoca al método* `Vaciar()` *para eliminar todos los* `nodos` *de la lista.*

35

Tarea 1.01.- DF Listas simples (constructor, vacía e iterador)

- Subir a MS Teams los archivos *.vsdx (Microsoft Visio) con diagramas de flujo de:
  - *Constructor de la lista simple*
  - *Propiedad para detectar si la lista simple está vacía*
  - *Iterador GetEnumerator()*

36

Diseño de la clase "roja"

- *Requisitos: Debe contener al menos un dato de los siguientes tipos:*
  - *Int*
  - *Double*
  - *String*
  - *Char*
  - *DateTime*
  - *Bool*
  - *String con la ruta del archivo que contiene una fotografía del objeto*

Se recomienda consultar las filminas
"El lenguaje C# y diseño de formas"
Para usar el PictureBox

37

Diseño de la clase "roja" (*cont.*)

IMPORTANTE

Considere un dato dentro de su clase "roja" que sea único e irrepetible que sirva para identificar a un objeto y que sea el criterio de comparación entre objetos

(Una clave para identificar y comparar objetos "rojos")

38

Tarea 1.02

- *Resolver la Tarea 1.02.- Diseño de la clase "roja" de la lista simple en MS Teams*
- *Subir el archivo *.vsxd con el diagrama de la clase "roja" elaborado en Microsoft Visio*
- *Incluir las interfaces correspondientes*

39

Ejercicio

- Hacer en clase el diagrama de flujo de:
  - *Método para agregar un objeto "rojo" a la lista*

40

Diseño de la forma de la aplicación visual

- *Requisitos: Debe contener al menos uno de estos controles visuales:*
  - *Textbox*
  - *Button*
  - *ComboBox*
  - *DateTimePicker*
  - *CheckBox*
  - *PictureBox*
  - *DataGridView*
  - *RadioButton*

Elija el control adecuado para cada dato capturado

Se recomienda consultar las filminas "Uso de los controles visuales"

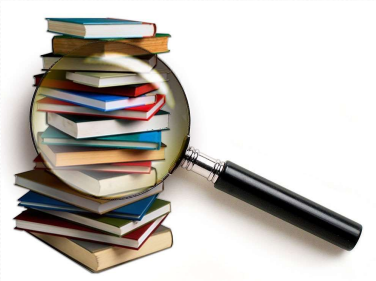
41

Recomendación de diseño de la forma

42

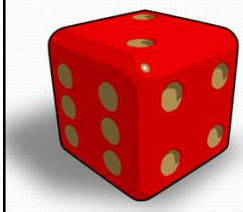
Investigación

- *Agregue un botón a su aplicación para crear `10` nodos con datos generados de manera aleatoria*
- *Muestre los datos generados en el dataGridView*



43

LECTURA



Para generar datos aleatorios, se recomienda la lectura de los apuntes

¿Cómo generar datos aleatorios en C#?

Incluye generar:

- *Nombres aleatorios*
- *Sexo*
- *Fecha*
- *Datos booleanos*
- *etc.*

https://nlaredo.tecnm.mx/takeyas/Apuntes/Fundamentos%20de%20Programacion/Apuntes/07.-Aleatorios.pdf

44

Tarea 1.03.- Diseño de la forma de listas simples

- Diseñar la forma en C#:

1. Declarar y crear un objeto global para la lista simple ordenada (objeto "verde")
2. Declarar y crear un objeto "rojo" local
3. Capturar los datos del objeto "rojo" usando los controles visuales adecuados
4. Agregar el objeto "rojo" a la lista simple
5. Recorrer los nodos de la lista simple ordenada para visualizar los datos de los objetos "rojos" en un dataGridView de solo lectura
6. Agregar un botón para generar 10 nodos con datos aleatorios

- *Subir a MS Teams un archivo comprimido con la aplicación completa (P. ejem. LopezTakeyasBruno.ZIP)*

45

Tarea 1.04.- DF Listas simples (Buscar)

- Subir a MS Teams el archivo *.vsdx (Microsoft Visio) con diagrama de flujo de:
  - *Método para buscar un objeto "rojo" en la lista simple (debe devolver el objeto encontrado)*

46

Eliminación de datos en una lista simple ordenada

- *Se recorre la lista para localizar el nodo con el objeto "rojo" que se desea eliminar*
- *Durante el recorrido es necesario "guardar" el nodo previo*
- *Se puede interrumpir la búsqueda por anticipado ya que la lista almacena datos ordenados*

Situaciones críticas de las bajas en una lista simple ordenada

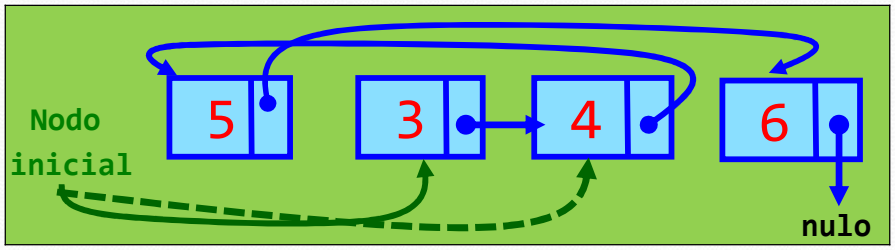
- *Baja a lista vacía.- Se dispara una excepción*
- *Baja al principio.- Se elimina el dato menor*
- *Baja intermedia.- Se elimina un dato ubicado entre otros*
- *Baja al final.- Se elimina el último dato*
- *Verificar la existencia del dato*
- *Devolver el objeto "rojo" eliminado*

48

Baja al principio en una lista simple ordenada

Cuando se elimina el primer nodo de la lista:

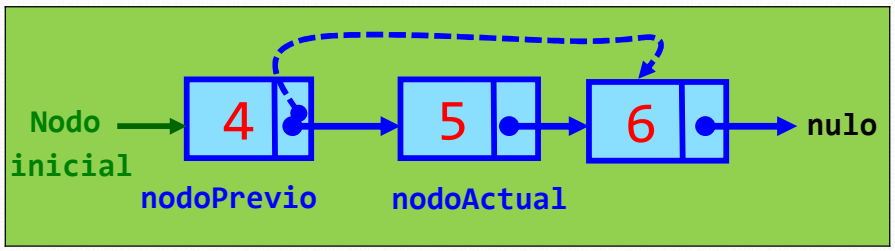
1) *El NodoInicial apunta al nodo que apuntaba el primer nodo de la lista*
2) *Se elimina el nodoActual*



Baja intermedia en una lista simple ordenada

Cuando se elimina un nodo intermedio:

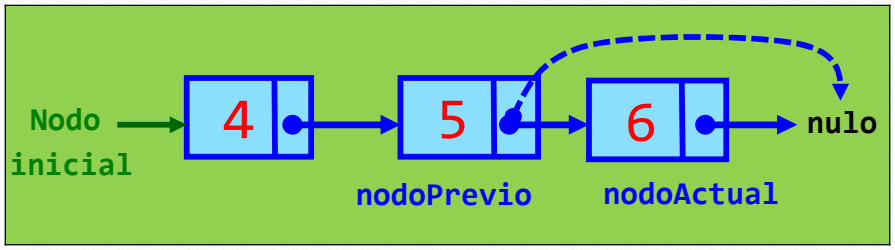
1) *El nodoPrevio apunta al nodo que apuntaba el nodoActual*
2) *Se elimina el nodoActual*



Baja al final en una lista simple ordenada

Cuando se elimina el último nodo de la lista:

1) *El nodoPrevio apunta al nodo que apuntaba el nodoActual (ahora apunta a nulo)*
2) *Se elimina el nodoActual*



Eliminación de objetos

- *La forma natural de borrar un objeto es asignarle el valor null*

```
// Creación del objeto "azul" del nodoActual
ClaseNodo<Tipo> nodoActual = new ClaseNodo<Tipo>();
   .
   .
nodoActual = null; // Eliminación del nodoActual
```

- *Sin embargo, no todos los datos aceptan el valor null, entonces... ¿cómo se eliminarían?*

52

Destructor de la clase "azul"

- *La `ClaseNodo<Tipo>` es parametrizada y está preparada para recibir un objeto "rojo" de cualquier tipo.*
- *El destructor de la clase "azul" elimina el objeto con datos "rojo" que contiene.*
- *Utiliza `default(Tipo)` para eliminar el objeto "rojo" porque desconoce si el ObjetoConDatos acepta el valor null.*

```
~ClaseNodo() // Destructor de la clase "azul"
{
     // Elimina el ObjetoConDatos "rojo"
     ObjetoConDatos = default(Tipo);
}
```

Tarea 1.05.- DF Listas simples (Eliminar)

- Subir a MS Teams el archivo *.vsdx (Microsoft Visio) con diagrama de flujo de:
  - Método para eliminar un objeto "rojo" de la lista
    - ***Devolver*** *el objeto "rojo" eliminado*

54

Operación del método para eliminar un nodo de la lista

- *Seleccionar un renglón del dataGridView y después oprimir el botón "Eliminar"*
- *Al seleccionar un renglón del dataGridView se deben mostrar los datos en los controles adecuados*
- *Confirmar la operación*
- ***Devolver** el objeto "rojo" eliminado*
- *Se debe actualizar el dataGridView después de la eliminación de un nodo*

55

Lectura

- *Se recomienda la consulta de las filminas*

El lenguaje C# y diseño de formas

https://nlaredo.tecnm.mx/takeyas/Apuntes/POO/Apuntes/03.-ElLenguajeCSharp.pdf

- *Para determinar: ¿Cómo seleccionar un renglón de un dataGridView?*

56

Vaciar la lista simple ordenada

- Situaciones críticas:
  - *Verificar si la lista ya está vacía*
  - *Recorrer los nodos "azules" de la lista*
  - *Durante el recorrido, guardar el* `nodoPrevio`
  - *Eliminar el* `nodoPrevio`
  - *El* `NodoInicial` *debe apuntar a nulo*

57

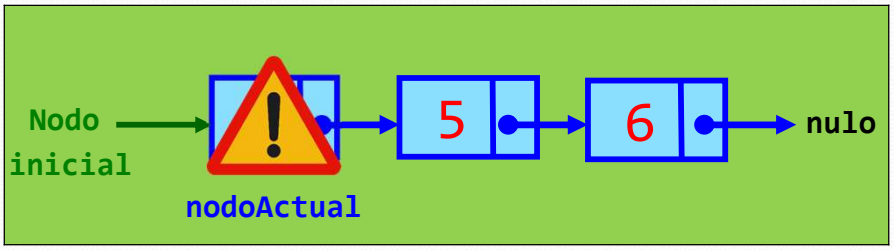
Precaución al vaciar la lista

- **NOTA IMPORTANTE:**
  - *Durante el recorrido de los nodos "azules" de la lista,* ***NO*** *se debe eliminar el* `nodoActual`
  - *Si se eliminara el* `nodoActual` *se pierde el apuntador* `Siguiente`
  - *Se perdería la secuencia lógica de los nodos "azules"*

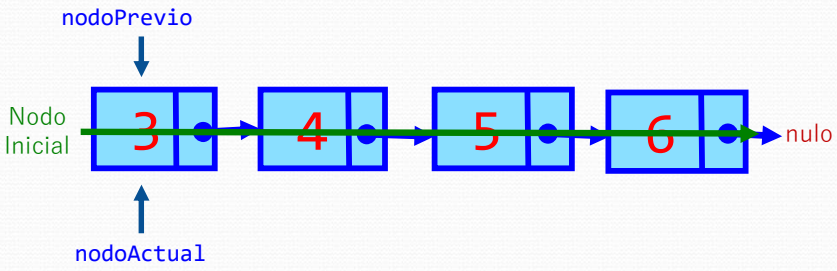
58

Precaución al vaciar la lista

- NOTA IMPORTANTE:
  - *Si se eliminara el `nodoActual` . . .*
  - *¿Cómo verificaría cuál es el siguiente nodo?*
  - *Por lo tanto **NO** debe eliminarse el `nodoActual` sino el `nodoPrevio`*



Ejemplo: Vaciar la lista



Termina el recorrido

60

Tarea 1.06.- DF Listas simples (Vaciar y Destructor)

- Subir a MS Teams los archivos *.vsdx (Microsoft Visio) con diagramas de flujo de:
  - *Método para vaciar la lista simple*
  - *Destructor de la lista simple*

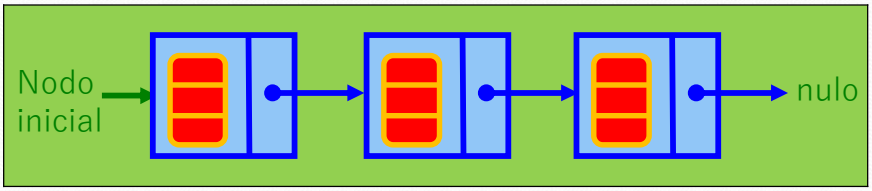
61

Tarea 1.07.- Aplicación completa de listas simples

- *Agregar un botón "Vaciar"*
- *Solicitar al usuario que confirme las operaciones (insertar, eliminar, vaciar, etc.). Preguntarle si está seguro que desea realizar la operación solicitada*
- *Mostrar los mensajes adecuados*
- *Subir a MS Teams un archivo comprimido con la aplicación completa (P. ejem. LopezTakeyasBruno.ZIP)*

62

Nivel de abstracción



- *No se debe perder de vista que los objetos "rojos" son privados, por lo tanto sus componentes son inacesibles para el objeto "verde"*

63

Nivel de abstracción (*cont.*)

- *El objeto "verde" recibe un objeto "rojo" y lo compara para almacenarlo ....*

  ¡¡¡ SIN SABER LO QUE TIENE DENTRO !!!

- *¿Cómo es posible que el objeto "verde" compare y almacene objetos "rojos" sin tener acceso a sus componentes?*

64

"Pensar en objetos …"

- *El objeto "verde" **NO** compara los objetos "rojos" (ellos mismos se comparan entre sí).*
- *El objeto "verde" **NO** requiere acceso a los componentes de los objetos "rojos" para manipularlos.*
- *Recibe cualquier tipo de objetos "rojos" …*

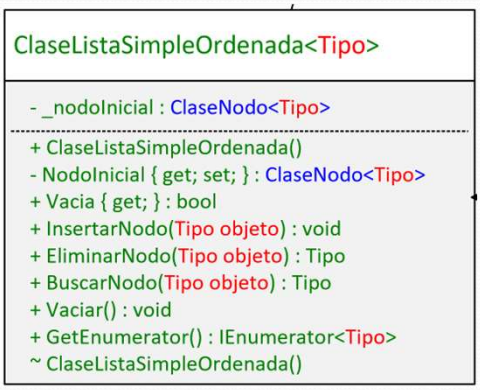
¡¡ SIN MODIFICAR NI UNA LÍNEA DE SU CÓDIGO !!!

- ***¿Cómo lo logra? …***
  - *Clases parametrizadas*
  - *Uso de interfaces*
  - *Composición*
  - *Comportamiento polimórfico*
  - *Restricción de tipos*

65

Diseño de una clase polimórfica

- La clase de la lista tiene estas características:
  - *Ordena los objetos "rojos"*
  - *No permite duplicados*



66

Diseño de una clase polimórfica *(cont.)*

- *¿Qué cambios se harían para que la misma clase "**verde**" almacene los datos desordenados?*
- *Pero …*
- *¡¡¡ Sin modificar ninguna línea de código de la clase "**verde**" !!!*

67

Diseño de una clase polimórfica *(cont.)*

- *¿Qué cambios se harían para que la misma clase "**verde**" permita objetos "rojos" duplicados?*
- *Pero …*
- *¡¡¡ Sin modificar ninguna línea de código de la clase "**verde**" !!!*

68

Diseño de una clase polimórfica *(cont.)*

- *¿Qué cambios se harían para que la misma clase "**verde**" muestre cualquiera de los siguientes comportamientos:*

- *Objetos "rojos" ordenados sin duplicados*
- *Objetos "rojos" ordenados con duplicados*
- *Objetos "rojos" desordenados sin duplicados*
- *Objetos "rojos" desordenados con duplicados?*

69

Ordenamiento lógico vs. ordenamiento físico

- *La lista simple mantiene un orden lógico de los datos*

- *El ordenamiento es controlado por medio de los apuntadores*

- *En otras estructuras de datos el ordenamiento es físico*

70

Ordenamiento lógico vs. ordenamiento físico (ejemplo)

- *Suponga que un grupo de estudiantes se incriben en una escuela. . .*
- *"**Rdz**" llegó primero, luego "**Glz**" y por último llega "**Mtz**"*

Arreglo	"Rdz"	"Glz"	"Mtz"
	0	1	2

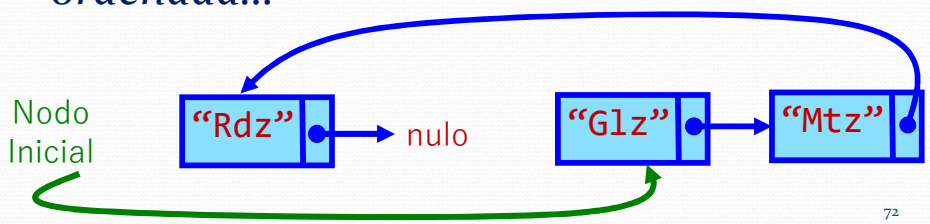
71

Ordenamiento lógico vs. ordenamiento físico (ejemplo)

- *Si se almacenaran en un arreglo...*

Arreglo	"Rdz"	"Glz"	"Mtz"
	0	1	2

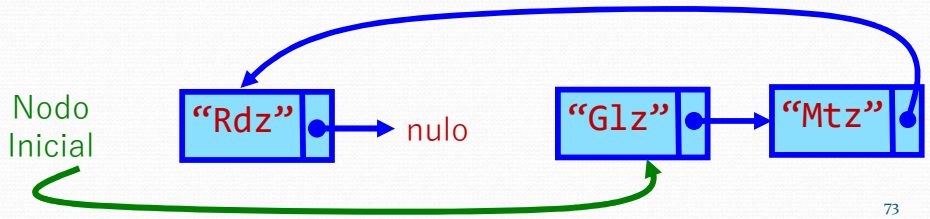
- *Si se almacenaran en una lista simple ordenada...*



72

Ordenamiento lógico vs. ordenamiento físico (conclusión)

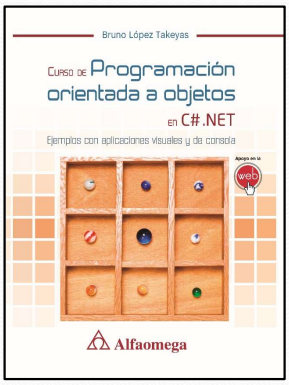
- *La lista simple mantiene ordenados alfabéticamente los estudiantes independientemente del orden en el que se inscribieron*

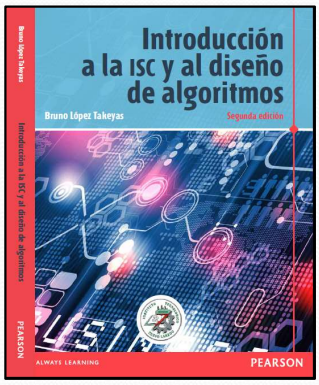


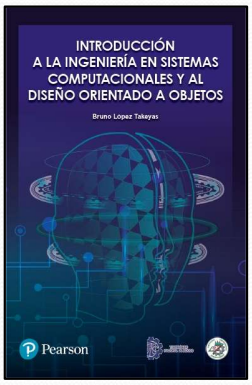
73

Otros libros del autor

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







bruno.lt@nlaredo.tecnm.mx

Bruno López Takeyas