Fundamentos de Sistemas de Informacion

Listas y Multilistas

Logro de sesion

Al finalizar la sesión, el estudiante describe las principales características de la estructura de datos tipo List. Participando activamente en las preguntas.

List

- Una lista es probablemente la segunda estructura de datos más utilizada.
- Los datos se encuentran dispersos por el RAM y se conectan a través de punteros.
- Lo único que se tiene es el puntero al primer elemento al igual que en el arreglo.
- La gran diferencia con un arreglo es que en una lista los datos **no** están ubicados de forma consecutiva en RAM a diferencia de los arreglos.
- En .NET se puede utilizar la clase List que ofrece todos los métodos para manipular una lista pero es necesario entender como funcionan estos métodos.

Clase List

Características:

- Define métodos para manipular, por índice, la inserción, búsqueda, ordenamiento de sus elementos.
- Permite almacenar objetos de cualquier tipo.

Clase List– Creando una Lista de elementos

Primero, asegurarse que se hace referencia a: System. Collections.Generic

```
List<tipo de dato> _lst=new List<tipo de dato>();
```

Es mejor especificar adecuadamente el tipo de dato para evitar que se realice typecasting en todo momento.

Clase List– Agregar un elemento

Para agregar un nuevo elemento a la colección debemos de utilizar el método **add**.

Ejemplo:

```
List<int> _lst=new List<int>();
int _variable=45;
_lst.Add(5);
_lst.Add(_variable);
```

Clase List– Insertar un elemento

Otra alternativa es usar el método **insert**. Pero se debe de especificar la posición.

Ejemplo:

```
List<int> _lst=new List<int>();
int _variable=45;
_lst.Insert(0,5);
_lst.Insert(1,_variable);
```

Clase List– Recorrido de la Lista

La forma más directa y fácil de recorrer una List es mediante un foreach

```
List<int> _lst = new List<int>();
foreach (int x in _lst)
        Console.WriteLine(x);
```

Clase List– Otros métodos importantes

- **Clear();**

  Elimina todos los elementos que contiene la lista

- **Contains(elemento);**

  Indica si un elemento se encuentra o no en la lista.

- **ConvertAll(_función de conversión)**

  Convierte los elementos de la lista a otro tipo. La función de conversión define la conversión de cada elemento.

Clase List– Otros métodos importantes

- **Count();**

  Retorna la cantidad de elementos de la lista.

- **Distinct();**

  Retorna una lista con los elementos, sin repetir, de una colección.

- **Element At(posición);**

  Retorna el elemento de la lista que se encuentra en la posición indicada.

Clase List– Otros métodos importantes

- **Remove(elemento);**

  Elimina el elemento indicado de la lista

- **RemoveAll(predicado);**

  Elimina todos los elementos que cumplen con el predicado

- **RemoveAt(posición);**

  Elimina el elemento de la lista que se encuentra en la posición indicada.

Clase List– Otros métodos importantes

- **Exists(predicado);**

  Retorna True o False, dependiendo si un elemento de la lista cumple con la condición dada.

  predicado : función delegado que define el criterio de búsqueda.

Clase List – Búsqueda de elementos

- **Find(predicado)**

Retorna el primer elemento que cumple con la condición dada en el predicado

- **FindAll(predicado)**

Retorna todos los elementos que cumplen con la condición dada en el predicado

- **FindIndex (predicado)**

Retorna el índice del primer elemento que cumple con la condición dada en el predicado

- **FindLast(predicado)**

Retorna el último elemento que cumple con la condición dada en el predicado.

Multilista

- Una multilista es simplemente una lista que contiene otra(s) lista(s).

- El concepto de multilista se aplica por lo general con listas, pero es posible mezclar vectores con listas para crear una multilista.

Lista de Cursos

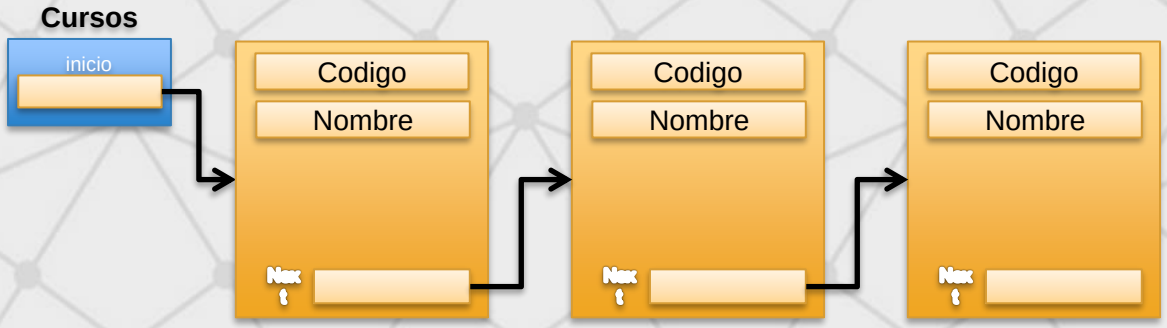
- Supongamos que necesitamos almacenar una lista con los cursos de la UPC.
- Cada curso contiene la siguiente información:
  - Código del curso
  - Nombre del curso
- Para resolver el problema nos conviene utilizar una clase que almacene estos dos valores:

```
class CCurso {
    public string Codigo;
    public string Nombre;
}
```

Relaciones entre clases

- Una vez que tenemos la estructura podemos crear una lista utilizando la clase List.

List<CCurso> cursos;



Lista de Alumnos

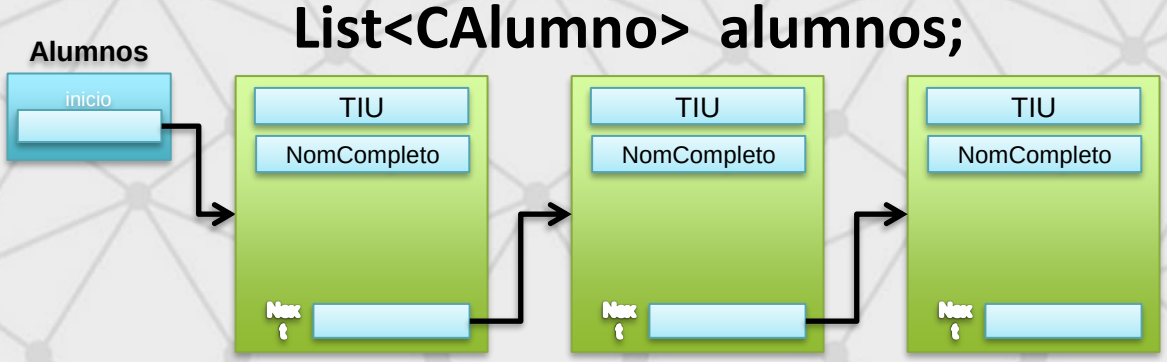
- Supongamos que necesitamos almacenar una lista con los alumnos de la UPC.
- Cada alumno contiene la siguiente información:
  - Código TIU
  - Nombre Completo

- Para resolver el problema nos conviene utilizar una estructura que almacene estos dos valores:

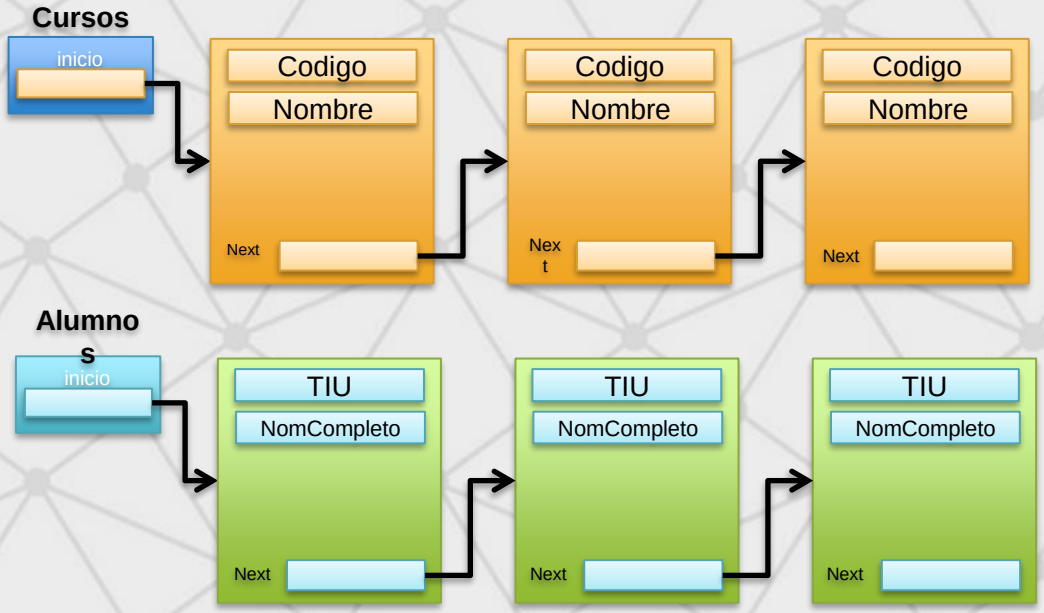
```
class CAlumno {
   public int TIU;
   public string NomCompleto;
}
```

Lista de Alumnos con .NET List

- Una vez que tenemos la estructura podemos crear una lista utilizando la clase List.



Cursos y Alumnos



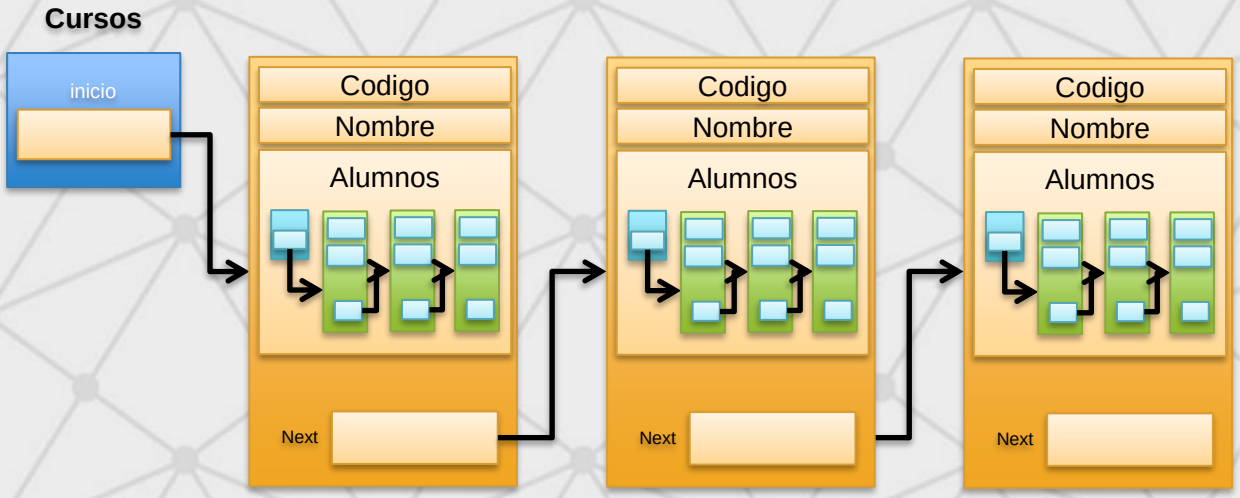
¿Como podemos saber cuáles son los alumnos de un curso específico?

Crear una multilista

- Para solucionar el problema anterior, lo que podemos hacer es agregar a cada curso una lista de alumnos.

```
class CCurso {
  public string Codigo;
  public string Nombre;
  public List<CAlumno> Alumnos;
}
```

Multilista



Ahora cada curso tiene un código, nombre y una lista de alumnos.

Ejercicios de implementación de multilistas

- Utilizando la clase List implementar los siguientes métodos:
  - Insertar Curso
  - Insertar Alumno En Curso
  - Listar Alumnos De Curso
  - Listar Cursos
  - Dame Curso Con Mas Alumnos
  - Curso Existe
  - Listar Cursos Que Lleva Un Alumno
  - Listar Cursos Que No Tienen Alumnos
  - Dame Alumno Con Numero TIU Mayor

Ejercicio para el laboratorio

- Implementar una clase CMatriculaUPC que permita llevar un control de los cursos y los alumnos que llevan cada uno de los cursos.
- La clase CMatriculaUPC deberá tener como mínimo todos los métodos descritos en la diapositiva anterior.
- Desarrolle una aplicación en entorno visual que permita administrar los cursos y los alumnos matriculados en cada curso. Considere que un mismo alumno se puede matricular en más de 1 curso. (Duplique al alumno en cada curso).