Equals, Listas y Contains

Listas

Es una estructura de datos que permite almacenar una serie de elementos y acceder a ellos mediante la posición. Está definida en el namespace System.Collections.Generic.

Las listas son tipadas y por lo tanto debemos definir el tipo de dato que van a almacenar, la sintaxis es la siguiente: List<T>, siendo T el tipo de datos.

Solo permite agregar a la lista tipos de datos que coincidan con el que la lista tiene definido.

En el caso de una lista que guarda números enteros:

```
List<int> listaNumeros = new List<int>();
```

En el caso de una lista que guarda objetos de tipo Cargo:

```
List<Cargo> cargos = new List<Cargo>();
```

En c# contamos un lenguaje - Language-Integrated Query - LINQ que permite recorrer listas y trabajar sobre ellas. **Es importante aclarar que en P2 NO SE PUEDE UTILIZAR**

Listas

Para poder trabajar con la lista es necesario inicializar la misma , ya que su valor por defecto es null.

Tenemos dos formas de inicializarla:

En la declaración:

```
private List<Cargo> cargos = new List<Cargo>();
```

En el constructor:

```
private List<Cargo> cargos;
public Sistema()
{
    cargos = new List<Cargo>();
}
```

Count y Add

Count: Retorna el largo de la lista.

Add(): Recibe por parámetro el elemento a agregar.

```
List<int> numeros = new List<int>();
numeros.Add(10);
numeros.Add(20);
Console.WriteLine( numeros.Count); //Largo 2
```

Equals

Equals es un método que permite comparar elementos.

Su respuesta es un valor booleano.

Los tipos de datos más comunes (string, int, double, etc) ya tienen implementado el equals, y por lo tanto alcanza con llamarlos directamente al momento de comparar.

```
string palabra = "Hola";

int num = 2;

if (palabra.Equals("Hola"))

if (num.Equals(1))
```

Equals con objetos

El método Equals es un método que retorna un dato bool que permite establecer si un objeto es igual a otro.

Por defecto el método Equals definido en Object compara referencias para saber si el objeto es el mismo.

Ejemplo:

```
class Program

static void main(string [] args){

Alumno a = new Alumno();

Alumno b = a;

if(a.Equals(b)){ //retorna true ambas variables referencian al mismo objeto}

}
```

Equals con objetos

Otro ejemplo:

```
class Program
public static void main(string [] args){
        Alumno a = new Alumno();
        Alumno b = new Alumno();
        if(a.Equals(b)){ //retorna false; las referencias son distintas}
}
```

Si se desea, se puede sobreescribir el método Equals en la clase correspondiente indicando qué atributo se desea indicar como único en el objeto; es decir no deberían de poder existir dos objetos con el mismo valor en ese atributo en particular.

Equals con objetos

Si los objetos Alumno tienen un numEstudiante que no se puede repetir podríamos sobreescribir el método Equals para que retorne true si el numEstudiante es el mismo.

Para ello debemos de escribir en la clase Alumno lo siguiente:

```
public override bool Equals(object obj)
    {
        var estudiante = obj as Alumno;
        return estudiante != null &&
               numEstudiante == estudiante.numEstudiante;
     }
```

Index of

Recibe un objeto por parámetro y retorna la posición de ese elemento en la lista o -1 si no lo encuentra.

El Index of necesita el Equals redefinido para poder encontrar la incidencia del objeto. Podemos crear un objeto auxiliar, asignarle el valor buscado en su atributo y luego salir a buscar si encontramos uno igual.

```
Empleado aux = new Empleado();

aux.NroFuncionario = numeroFuncionario;

int i = listaEmpleados.IndexOf(aux);
```

Contains

Recibe un objeto por parámetro y devuelve true o false según se encuentre o no en la lista.

El Contains también necesita el Equals redefinido para poder encontrar la incidencia del objeto.

```
Empleado aux = new Empleado();
aux.NroFuncionario = numeroFuncionario;
bool i = listaEmpleados.Contains(aux);
```

Clear

Clear(): borra el contenido de la lista. Lleva el Count a 0

Filtrado

Recorrer cada elemento de la lista que deseo filtrar y chequear si cumple la condición deseada para agregarlo a una nueva lista filtrada.

```
1 reference
public List<Cargo> CargosPorNombre(string nombre)
{
    List<Cargo> aux = new List<Cargo>();
    foreach (Cargo item in _cargos)
    {
        if (item.Nombre.ToLower().Contains(nombre.ToLower()))
        {
            aux.Add(item);
        }
    }
    return aux;
}
```

Búsqueda

Recorrer cada elemento de la lista y si coincide con la búsqueda devuelvo una referencia al objeto o el valor null. Teniendo una referencia al objeto luego podrá ser modificado.

```
5 references
public Cargo BuscarCargoPorID(int id)
{
    int posicion = 0;
    Cargo cargo = null;
    while (cargo == null && posicion < _cargos.Count)
    {
        Cargo unCargo = _cargos[posicion];
        if (unCargo.Id == id)
        {
            cargo = unCargo;
        }
    }
    return cargo;
}
```

Ordenar

Sólo podrá ser realizado mediante interfaces para tal fin (IComparable, IComparer) a través del método Sort.