Clase 18

Lunes, 16 Octubre 2023

Indice

Instalación de paquetes Nuget

- Primero necesitamos una carpeta con nuestros archivos 'Nuget'. En este caso, necesitamos el 'UDK'
- Descomprimimos y borramos el archivo 'macOS'.
- No es conveniente cambiar el nombre del '6.0'.
- Ahora abrimos el Visual Studio y creamos un proyecto nuevo.
- Buscamos en el explorador de la derecha el archivo de nuestra solución y pulsamos el botón derecho. Buscamos la opción de 'administrar paquetes nuget'.
- En la pestaña de instalado no debería parecer ningún archivo.
- Pulsamos la rueda de opciones arriba a la derecha, para entrar en la pantalla donde añadiremos una nueva dirección nuget.
- Es importante **no borrar** los archivos ya existentes. Esto causa serios problemas. En lugar de eso, escogemos un nuevo nombre y examinamos nuestro ordenador para añadir nuestra carpeta propia de 'nuget'
- Ahora examinamos en la pantalla de administrador y seleccionamos los paquetes suministrador. Instalamos.
- En nuestro caso necesitamos los siguientes paquetes: **UDK** y **OpenAl.Soft version 1.19.1**
- Instalamos los dos paquetes y ahora ya podemos usar nuestro framework gráfico.
- En caso que el IDE no reconozca los nuevos 'nuget' instalados, debemos borrar la cache de los existentes. Esta opción se encuentra en /Herramientas/Opciones/Administrador de paquetes nuget/Borrar todas las caches de nuget.

Continuación clase Delfín

- Creamos con la interfaz gráfica un videojuego de Policías y ladrones.
- La clase **'Character'** diseña *un único personaje.*
- Añadimos un 'public enum' para definir el tipo de personaje que se crea con cada instancia: policia o ladrón.
- Es bueno que los 'enum' sigan la guía de estilo Java y se escriban con Mayúsculas
- El código es el siguiente:

```
'
public enum CharacterType
{
    POLICE,
    THIEF
}
public class Character
{
    // define la instancia de un ÚNICO Character
}
'
```

- NOTA: un string es una cadena de texto. El tipo 'string' proviene de la clase 'String' que es invisible para el usuario.
- un 'string' es realmente un 'String[]'. Es el *runtime* quien se encarga de simplificar su funcionamiento.
- El warning de un posible valor 'null' se quita marcando el tipo con el signo de interrogante '?'
- Creamos un personaje y le damos atributos

```
'
public class Character
{
    public string name;
    public CharacterType type;
}
public class Program
{
    Character c1 = new Character();
    Character c2 = c1;
    // En este momento solo hay UN personaje y DOS variables apuntando al mismo personaje.
    c1.name = "Poli1";
    c1.type = CharacterType.POLICE;

    // Creo una lista de Character
    List<Character> list;
    list = new List<Character>();
    list.Add(c1);
}
'
```

- **IMPORTANTE:** Si apuntamos todas las referencias al objeto hacia 'NULL' se destruye el objeto: c1 = null; c2 = null; entonces se destruye el objeto.
- **IMPORTANTE:** Si hemos añadido un objeto a la lista, el objeto se mantiene dentro de la lista.
- Cada objeto de una lista es un número que apunta a una dirección de la memoria. Las variables simplemente apuntan hacia ese número almacenado.

```
'
public class Program
{
    list.Add(new Character());      // se crea en la posicion 1
    list[1].name = "Ana";
    list[0].name = list[1].name;    // Ahora tengo dos objetos
    list.add(list[0]);              // La lista tiene tres objetos, pero hay dos que apuntan a la misma referencia
}
'
```

Funciones de objeto

- **FUNCION:** Se le pasa una lista de personajes y un string. Quiere saber si hay algun personaje que tenga ese nombre.

```
public static bool ContainsName(List<Character> list, string name)
{
    if( list == null || list.Count == 0)
        return null;

    for(int i = 0; i < list.Count; i++)
    {
        if(list[i].name == name)
        {
            return true;
        }
    }
    return false;
}
```

- En Java no se permite comprobar que dos strings son semejantes '=='
- **FUNCION:** Le paso una lista de Character y un string y me devuelve el primer personaje que coincida.

```
public static Character ContainsCharacter(List<Character> list, string name)
{
    if( list == null || list.Count == 0)
        return null;

    for(int i = 0; i < list.Count; i++)
    {
        if(list[i].name == name)
        {
            return list[i];
        }
    }
    return false;
}
```

- **FUNCION:** Le paso una lista de Character y me devuelve un 'true or false' si existe un duplicado.

```
public static bool ContainsDuplicate(List<Character> list, string name)
{
    if( list == null || list.Count == 0)
        return null;

    for(int i = 0; i < list.Count - 1; i++)
    {
        for(int j = i + 1; j < list.Count; j++)
        {
            if(list[i] == list[j]))
            {
                return true;
            }
        }
    }
    return false;
}
```

- Es aconsejable definir exactamente que representa un duplicado, ya que puede referirse a dos instancias con exactamente el mismo valor de sus **Atributos**, o también puede referirse a dos variables que apuntan a la misma instancia y por tanto, existe un único objeto con una **referencia duplicada.**