SERIALIZACIÓN

SERIALIZACIÓN CONCEPTOS GENERALES

Persistencia

- Se define por persistencia en el mundo de la POO, como la capacidad que tienen los objetos de sobrevivir al proceso padre que los creo. Esto decir, que su ciclo de vida excede de la del programa que lo instanció.

- La persistencia permite al programador almacenar, transferir y recuperar fácilmente el estado de los objetos.

¿Cómo Podemos Conseguir La Persistencia?

- La forma más común de conseguirlo es mediante la serialización.

SERIALIZACIÓN CONCEPTOS GENERALES

¿A Qué Serializaremos Una Clase?

- Lo más común es:
  - Hacerlo a un simple flujo de bytes en binario ya sea a disco o a memoria.
  - A la definición de algún lenguaje XML.
  - A alguna notación de objetos estándar como JSON.
- Aunque la mayoría de lenguajes ya la traen implementada. Si tuviéramos definir nosotros un interfaz OO para serializar objetos. Podría ser algo parecido a esto...

```
public interface ISerializacion<T>
{
    void Serializa(T dato, Stream flujo);
    T Deserializa(Stream flujo);
}
```

A la hora de serializar una clase llamaríamos a su método Serializa y este a su vez a los Serializa de los objetos y tipos que contenga, así sucesivamente.

SERIALIZACIÓN EN C#

- Muchos lenguajes como Java o C# solucionan la serialización de forma sencilla, ya que al serializar un objeto contenedor, este a su vez serializa mediante un mecanismo de reflexión y de forma trasparente aquellas referencias a objetos que contiene. Lo mismo sucede al cargar o deserializar un objeto.
- Para ello C# me ofrece el marcar mis clases como serializables a través de un atributo.

Atributos En .NET

- Un Atributo en .NET es una etiqueta de la sintaxis [nombre] que podremos aplicar a un ensamblado, clase, constructor, enumeración, campo, interfaz, método, etc… y que genera información en el ensamblado en forma de metadatos heredando de la clase Attribute.
- Por ejemplo si queremos realizar una simple serialización binaria etiquetaremos la clase a serializar y todas las que contenga con el atributo ya definido [Serializable], sobre la definición de la clase.

SERIALIZACIÓN EN C#

Serialización A Binario

- Para posteriormente serializar el tipo deberemos utilizar un formateador, el más común es el IFormatter, que se utiliza de la siguiente manera:

```
IFormatter <nombreobjeto> = new BinaryFormatter();

<nombreobjeto>.Serialize(<medioalmacenamiento>, <objetoaserializar>);
```

```
[Serializable]
public class MiClase
{
   public int n1;
   public int n2;
   public String str;
}
```

```
MiClase obj = new MiClase();
obj.n1 = 1;
obj.n2 = 24;
obj.str = "Some String";

IFormatter formatter = new BinaryFormatter();
using(Stream stream = new FileStream(
                          "MyFile.bin",
                          FileMode.Create,
                          FileAccess.Write))
{
   formatter.Serialize(stream, obj);
}
```

SERIALIZACIÓN EN C#

Deserialización De Binario

- De forma análoga realizaremos la deserialización.

```
IFormatter <nombreobjeto> = new BinaryFormatter();

Objeto = <nombreobjeto>.Deserialize(<medioalmacenamiento>);
```

```
IFormatter formatter = new BinaryFormatter();
MiClase obj = null;
using(Stream stream = new FileStream(
                    "MyFile.bin", FileMode.Open, FileAccess.Read))
{
     obj = formatter.Deserialize(stream) as MiClase;
}

if (obj != null) {
     Console.WriteLine("n1: {0}", obj.n1);
     Console.WriteLine("n2: {0}", obj.n2);
     Console.WriteLine("str: {0}", obj.str);
}
```

SERIALIZACIÓN EN C#

Serialización A Binario

- Una clase a menudo contiene campos que no se deben serializar. Por ejemplo campos específicos que almacenen datos confidenciales. Podremos excluir dichos campos del proceso aplicándoles el atributo [NonSerialized].

```
[Serializable]
public class MiClase
{
     public int n1;
     [NonSerialized]
     public int n2; // No entrará en el proceso de serialización.
     public String str;
}
```