**Estructura de datos II**

Tema: Serialización

Alumno: Juan Pablo Padilla Martin

**Introducción**

En este ensayo investigaremos el concepto de serialización tanto en lenguajes con los que hemos trabajado la materia como C o C++ como en otros aparte como java, primero haremos énfasis en la definición y los conceptos principales para después partir en el desarrollo de cada uno tratando de abarcar el máximo posible de información sobre dicho tema para después proceder a mostrar algunos ejemplos

**Serialización**

La serialización es un proceso mediante el cual podemos convertir objetos de un programa en ejecución en flujos de bytes capaces de ser almacenados en dispositivos, bases de datos o de ser enviados a través de la red y, posteriormente, ser capaces de reconstruirlos en los equipos donde sea necesario. Al hablar de “objetos”, no me refiero únicamente al significado que estos tienen en la POO; un objeto es cualquier estructura de datos, función o método que esté en memoria.

Uno de los principales objetivos de la serialización es permitir crear flujos de bytes independientes de la arquitectura de los equipos en los que se utilicen, inclusive, que los objetos puedan ser reconstruidos en otros programas sin importar que el lenguaje con el que son escritos sea diferente al que se usó para crear el objeto originalmente.

La importancia que tiene este procedimiento no se limita al envío de bytes por la red. Son varias las razones por las que muchos lenguajes de programación como C, C++, Java, C#, Python y Perl (entre otros) han incluido paquetes, módulos o API enfocadas en serializar datos.

Si el objeto que estamos intentando serializar contiene apuntadores a otros objetos o datos en memoria, el proceso de serialización puede volverse un poco -o muy- complicado porque, a pesar de que enviemos el objeto, es probable que las direcciones a las que hacen referencia los apuntadores no sean las mismas en los equipos donde se reconstruya, ni siquiera hay garantía de que hayamos enviado los objetos a los que referenciaban.

Para serializar un objeto, necesita el objeto que se va a serializar, una secuencia que contenga el objeto serializado y un Formatter. System.Runtime.Serialization contiene las clases necesarias para serializar y deserializar objetos.

Aplique el atributo SerializableAttribute a un tipo para indicar que se pueden serializar instancias de este tipo. Si se intenta serializar, pero el tipo no tiene el atributo SerializableAttribute, se produce una excepción.

Si no quiere que un campo de la clase sea serializable, aplique el atributo NonSerializedAttribute. Si un campo de un tipo serializable contiene un puntero, un controlador o alguna otra estructura de datos específica para un entorno concreto y el campo no se puede reconstituir correctamente en un entorno diferente, puede convertirlo en no serializable.

Si una clase serializada contiene referencias a objetos de otras clases marcadas como SerializableAttribute, esos objetos también se serializarán.

**Serialización básica y personalizada**

La serialización puede realizarse de dos formas, es decir, de manera básica y también personalizada. La serialización básica utiliza .NET Framework para serializar automáticamente el objeto.

**Serialización básica**

El único requisito de la serialización básica es que el objeto tenga aplicado el atributo SerializableAttribute. NonSerializedAttribute puede usarse para impedir la serialización de campos específicos.

Cuando se usa la serialización básica, el control de versiones de objetos puede causar problemas. Use la serialización personalizada si los problemas de control de versiones son importantes. La serialización básica es la manera más fácil de realizar la serialización, pero no proporciona mucho control sobre el proceso.

**Serialización personalizada**

En la serialización personalizada, puede especificar exactamente qué objetos se serializarán y cómo se llevará a cabo la serialización. La clase debe marcarse como SerializableAttribute e implementar la interfaz ISerializable.

Si desea que el objeto se deserialice también de forma personalizada, debe usar un constructor personalizado.

**EJEMPLO:**

Representación gráfica simple: