**LAS COLECCIONES GENERICAS**

A diferencia de las colecciones de datos tratadas en los post anteriores, las ***colecciones genéricas*** se encuentran en el namespace System.Collections.Generics ya que son colecciones con la misma funcionalidad que las colecciones no genéricas (las normales, las que están en el namespace System.Collections), con la diferencia que estas están orientadas a trabajar con un tipo de dato especifico.

**¿Con un tipo de dato especifico?** Si así es, porque a diferencia de las *colecciones no genéricas* en las que se pueden agregar **cualquier tipo de elementos** y todos son convertidos a System.Object, en este tipo de colecciones **solo es posible** almacenar elementos de un tipo de dato específico.

Definitivamente, una ***colección genérica*** solo puede almacenar datos de un tipo, **pero dicho tipo de dato es definido por nosotros mismos al momento de declararla e instanciarla**.

Es decir, no hay una clase para diferenciar las colecciones que operan con enteros de las que operan con cadenas de texto ni de las que operan objetos de tipo Empleado (por ejemplo). Absolutamente todas están definidas en la misma clase, lo único que las diferencia es la manera en que se instancia y se declara cada una de ellas. Ese es el secreto del ser genérica. Para entender mejor el concepto de la colección genérica, veamos el siguiente ejemplo:

[](https://developmania.files.wordpress.com/2008/10/clip-image002.jpg)

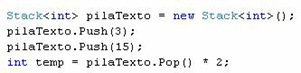
En el ejemplo se observa la implementación de la colección genérica List, que es la versión genérica del ArrayList tratado en un post anterior. Se puede deducir que la primera colección permitirá almacenar únicamente datos de tipo int, que la segunda almacenara solo cadenas de texto y que la última almacenara objetos de tipo TablaDatos, el cual es un tipo definido en una clase aparte.

En las definiciones de las tres colecciones se utilizó la clase List, es decir, que hay una clase “genérica” capaz de trabajar con el tipo de dato que indiquemos al momento de inicializarla. (Realmente no es cualquier tipo de dato, sino los tipos de datos soportados por la clase).

La diferencia para inicializar una clase genérica de una clase no genérica, es que en la primera se debe utilizar un parámetro adicional después del nombre de la clase y entre los caracteres < y >. Este parámetro debe ser un tipo de datos, que indica el tipo de datos con el que la colección deberá operar.

Así como está la colecciones List<T>, que es la versión genérica del ArrayList, también existen las colecciones genéricas Stack<T>, Queue<T> y Dictionary<T>, entre otras que son las versiones genéricas de las clases Stack, Queue y HashTable respectivamente.

Un ejemplo de la clase genérica de Stack seria:

[](https://developmania.files.wordpress.com/2008/10/clip-image004.jpg)

Se puede ver como se define una pila en la que únicamente se pueden almacenar datos de tipo entero, lo cual me permite obtener datos de la colección y tratarlos directamente como enteros sin necesidad de hacer algún tipo de cast, lo cual no sería posible con las colecciones no genéricas ya que el elemento es almacenado como un System.Object.

El hecho de que las colecciones genéricas operen con un tipo de dato definido en el momento de su declaración, las hace mucho más eficientes, ya que evita tener que realizar boxing, unboxing y casting de objetos. Adicionalmente los desarrolladores, obtienen un mayor grado de control sobre la información almacenada en la colección ya que en caso de intentar ingresar un objeto de un tipo de dato diferente al establecido en la inicialización de la colección, se arroja una excepción.

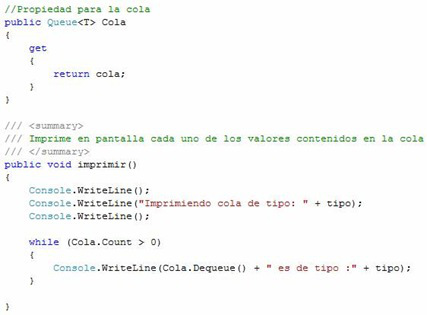
Además de esta ventaja que nos brindan las colecciones genéricas, tenemos la posibilidad de crear nuestras propias clases y métodos genérico cuyo comportamiento estará ligado al tipo de dato que se utilice al utilizarlo tal y como sucede con las colecciones. Para lograr esto debemos hacer uso de los “type parameters”, los cuales permiten establecer un parámetro cuyo tipo se desconocerá hasta que se indique en la declaración.

Para comprender mejor el funcionamiento de los “type parameters” que nos permiten crear clases genéricas, he desarrollado una aplicación de muestra que consiste en una clase impresora “genérica”, que es capaz de almacenar en una cola, objetos de un tipo específico (del tipo que se define al momento de definir un objeto de esta clase). Adicionalmente expone un método “imprimir” que se encarga de imprimir en pantalla cada uno de los valores contenidos en la cola. A continuación, presento los fragmentos principales del código.

[](https://developmania.files.wordpress.com/2008/10/clip-image006.jpg)

En esta primera imagen, se observa como se ha definido una clase Impresora genérica. Note que en este caso se utiliza la letra T para representar el “type parameter”, sin embargo, es posible utilizar cualquier letra o palabra. Mediante este parámetro estamos indicando que al momento de definir e instanciar un objeto de tipo Impresora, se debe pasar como parámetro entre < y > el tipo de datos con el que se desea trabajar.

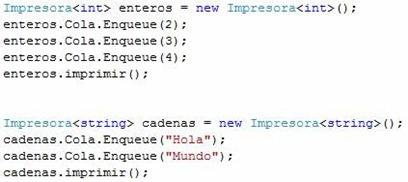
En el constructor de esta clase, se puede ver como la Queue genérica que tiene la clase Impresora como atributo toma su mismo tipo de dato, es decir, si al crear la clase Impresora indicamos el tipo de dato int, la Queue también tomara el tipo de dato int.

[](https://developmania.files.wordpress.com/2008/10/clip-image008.jpg)

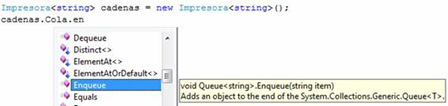
En esta segunda imagen se observa una propiedad que retorna una cola genérica, pero el type parameter para indicar que es una Queue genérica es el mismo utilizado en nuestra clase Impresora, lo cual indica que retornara una Queue del tipo de dato utilizado en la clase.

Finalmente se observa el método imprimir, cuyo objetivo es indicar el tipo de Queue que se esta imprimiendo y además mostrar cada uno de los elementos que contiene.

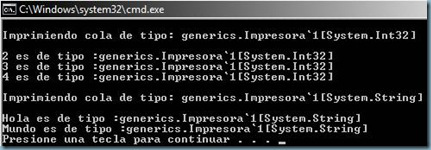
Como paso final, para comprobar el funcionamiento de la clase Impresora, se crea un objeto de tipo Impresora de enteros, así:

[](https://developmania.files.wordpress.com/2008/10/clip-image010.jpg)

Otra de las ventajas que nos brinda la utilización de Generics, es que los “type parameters” se traducen en tiempo de compilación, lo cual implica que es posible darnos cuenta en el momento que desarrollamos nuestra aplicación, el tipo de datos con el que se esta trabajando en cada instancia de la clase Impresora, como se ve a continuación:

[](https://developmania.files.wordpress.com/2008/10/clip-image012.jpg)

Finalmente, se observa el resultado de la ejecución de la aplicación en consola:

[](https://developmania.files.wordpress.com/2008/10/clip-image014.jpg)

En conclusión, debemos utilizar las colecciones genéricas tanto como sea posible en lugar de utilizar las colecciones no genéricas, ya que las primeras incrementan el rendimiento de la aplicación ya que evita un sin numero de casting, boxing y unboxing lo que permite realizar un mejor manejo de la memoria y procesamiento.

Además podemos crear propias clases y/o métodos genéricos para aumentar la flexibilidad y la posibilidad de reutilizar nuestro código.