Universidad del Istmo de Guatemala
Facultad de Ingeniería
Ing. en Sistemas
Informática 2
Dylan Gabriel Rodas Samayoa - rodas171315@unis.edu.gt

Hoja de Trabajo #4

17 de Marzo, 2018

Ejercicio #4:

Ventajas del método Head con Genéricos

Al permitirle especificar los tipos concretos sobre los que actúa una clase o método genérico, la característica de genéricos traspasa la carga de la seguridad de tipos al compilador. No hay ninguna necesidad de escribir código para comprobar que el tipo de datos es correcto, porque esto se hace en tiempo de compilación. Se reduce la necesidad de convertir los tipos y la posibilidad de que se produzcan errores en tiempo de ejecución.

Los genéricos proporcionan seguridad de tipos sin el trabajo extra de realizar de varias implementaciones. No hay necesidad de heredar de un tipo base y reemplazar los miembros. Además de la seguridad de tipos, los tipos de colección genéricos suelen conseguir mejor rendimiento para almacenar y manipular tipos de valor porque no hay necesidad de realizar conversiones de los tipos de valor.

Los delegados genéricos habilitan las devoluciones de llamada con seguridad de tipos sin la necesidad de crear varias clases de delegado. Los delegados genéricos también se pueden utilizar en código generado dinámicamente sin que sea necesario generar un tipo de delegado. Esto aumenta el número de escenarios en los que puede utilizar métodos dinámicos ligeros en lugar de generar ensamblados completos.

En muchos casos, los compiladores de Visual Basic, Visual C++ y C# pueden determinar a partir del contexto los tipos que se utilizan en una llamada de método genérico, lo que simplifica enormemente la sintaxis cuando se utilizan métodos genéricos. Por ejemplo, en el código siguiente se puede observar los beneficios de una implementación con genéricos:

```
class EjemploLatex
{
    public static object Head(object[] arreglo)
    {
        return arreglo[0];
    }

    public static T HeadGenerico<T>(T[] arreglo)
    {
        return arreglo[0];
```

```
    }

    public static void Prueba()
    {
        //Esto provoca un error en tiempo de compilacion, debido a una excepcion
        //de casteo no valido por intentar castear una cadena a un entero.
        string[] valoresStr = new string[];
        valoresStr.Add("Enero");
        valoresStr.Add("Diciembre");
        int resultado = (int)Head(valoresStr);

        //Pero el metodo generico, permite mantener un control del tipo de dato
        //que se obtendra y no hay necesidad de boxing, ni casteo.
        int[] valoresInt = new int[];
        valoresInt.Add(2017);
        valoresInt.Add(2018);
        int resultadoInt = HeadGenerico(valoresInt);
    }
}
```