**Práctica 1**

1. Crear un proyecto de Windows: “TestGeneric”
2. Agregar una carpeta en TestGeneric, nombre: “TiposGenericos”
3. Agregar una clase en TiposGenericos, nombre: “Operaciones”
4. Crear un método genérico “intercambio” según los siguientes ejemplos de código definidos a continuación:

static void intercambio(ref decimal a, ref decimal b)

{

decimal t = a;

a = b;

b = t;

}

static void intercambio (ref int a, ref int b)

{

int t = a;

a = b;

b = t;

}

1. En un formulario diseñar una simple interface y probar el método genérico con parámetros decimales y luego con parámetros enteros.

**Práctica 2**

* Revisar el siguiente ejemplo y crear una clase Generica “Domain” para el siguiente caso:
* Considerar solo las clases abstractas para crear la clase generica

**public abstract class IntegerIdDomain**

**{**

**public int Id { get; }**

**public IntegerIdDomain(int id)**

**{**

**Id = id;**

**}**

**}**

**public abstract class StringIdDomain**

**{**

**public string Id { get; }**

**public StringIdDomain(string id)**

**{**

**Id = id;**

**}**

**}**

**public abstract class GuidIdDomain**

**{**

**public Guid Id { get; }**

**public GuidIdDomain(Guid id)**

**{**

**Id = id;**

**}**

**}**

// -------------test

public class Customer : IntegerIdDomain

{

    public Customer(int id, string name) : base(id)

    {

        Name = name;

    }

    public string Name { get; }

}

public class Order : GuidIdDomain

{

    public Order(Guid id, string address) : base(id)

    {

        Address = address;

    }

    public string Address { get; }

}

public class Shipment : StringIdDomain

{

    public Shipment(string id, string meansOfTransportation) : base(id)

    {

        MeansOfTransportation = meansOfTransportation;

    }

    public string MeansOfTransportation { get; }

}

* Realizar una prueba de la clase genérica Domain para Customer, Order y Shipment.

Práctica 3

1-Agregar una aplicación de cosole, nombre: “ConsoleGeneric”

2-Agregar una clase “Generator” y definir los siguientes métodos:

|  |
| --- |
| using System;  public class Generator  {  // Display the value of an integer  public void Show(int value)  {  Console.WriteLine(value);  }  // Display the value of a double-precesion value  public void Show(double value)  {  Console.WriteLine(value);  }  // Display the value of a character  public void Show(char value)  {  Console.WriteLine(value);  }  }  3- Probar la clase Generator desde una consola o formulario en windows |

|  |
| --- |
| public class Exercise  {  static int Main()  {  var exo = new Generator();    // Call the version of the method that displays an integer  exo.Show(246);    // Call the version of the method that displays a character  exo.Show('G');    // Call the version of the method that displays a decimal  exo.Show(355.65);    return 0;  }  } |

public class Exercise

{

static int Main()

{

var exo = new Generator();

// Call the version of the method that displays an integer

var Value1 = 246;

exo.Show(Value1);

// Call the version of the method that displays a character

var Value2 = 'G';

exo.Show(Value2);

// Call the version of the method that displays a decimal

var Value3 = 355.65;

exo.Show(Value3);

return 0;

}

}

4- Agregar una nueva clase,nombre: “GeneratorGenric”, aplicar generic analizando los métodos de sobrecarga escritos en el paso anterior.

5- Realizar la prueba de la clase GeneratorGenric.

Práctica 4

1. Agregar una aplicación de consola, Nombre: **CommercialStore1**
2. Para crear una nueva clase, nombre de **StoreItem**
3. Cambiar el archivo de la siguiente manera:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace CommercialStore1

{

public class StoreItem

{

public class CItem

{

public double Item;

public CItem Next;

}

public CItem Head = null;

public int Size;

public int Count()

{

return Size;

}

public int Add(CItem NewItem)

{

CItem Sample = new CItem();

Sample = NewItem;

Sample.Next = Head;

Head = Sample;

return Size++;

}

public CItem Retrieve(int Position)

{

CItem Current = Head;

for (int i = Count() - 1; i > Position && Current != null; i--)

{

Current = Current.Next;

}

return Current;

}

public void ShowItem(double item)

{

Console.WriteLine("Item: {0}", item);

}

}

}

1. Realizar una prueba

|  |
| --- |
| using System;  namespace CommercialStore1  {  public class Program  {  static void Main()  {  StoreItem exo = new StoreItem();  StoreItem.CItem Part;  Part = new StoreItem.CItem();  Part.Item = 97.43;  exo.Add(Part);  Part = new StoreItem.CItem();  Part.Item = 274.87;  exo.Add(Part);  Part = new StoreItem.CItem();  Part.Item = 8.7873;  exo.Add(Part);  Part = new StoreItem.CItem();  Part.Item = 2764.4;  exo.Add(Part);  Part = new StoreItem.CItem();  Part.Item = 92.4662;  exo.Add(Part);  Part = new StoreItem.CItem();  Part.Item = 66800.85;  exo.Add(Part);  Console.WriteLine("-=- List of Items -=-");  for (int i = 0; i < exo.Count(); i++)  {  StoreItem.CItem One = exo.Retrieve(i);  exo.ShowItem(One.Item);  }  Console.WriteLine("\nNumber of Items: {0}\n", exo.Count());  }  }  } |

1. Resultado

|  |
| --- |
| -=- List of Items -=-  Item: 97.43  Item: 274.87  Item: 8.7873  Item: 2764.4  Item: 92.4662  Item: 66800.9  Number of Items: 6  Press any key to continue . . . |

**Explicación**

|  |
| --- |
| **Método Genérico Creación** |
|  |

Un método genérico es un método que puede procesar un valor cuyo tipo se conoce sólo cuando se accede a la variable. Para crear un método genérico, en el lado derecho del nombre del método, escriba el operador <>. Dentro de este operador, introducir una letra o un nombre, lo que se conoce como tipo de parámetro. Aquí hay un ejemplo:

public class Generator

{

public void Show<TypeOfValue>()

{

}

}

Una de las formas en que puede utilizar el tipo de parámetro es pasar un argumento al método. Haga esto precediendo al nombre del argumento con el tipo de parámetro. Aquí hay un ejemplo:

public class Generator

{

public void Show<TypeOfValue>(TypeOfValue value)

{

}

}

En el cuerpo del método, puede procesar el argumento como mejor le parezca. Como mínimo, y basándonos en nuestro programa anterior, puede simplemente mostrar el valor pasándolo al método Console.WriteLine (). Aquí hay un ejemplo:

public void ShowItem<T>(T item)

{

Console.WriteLine("Item: {0}", item);

}

**Preparar la clase Generator para usar genéricos**

using System;

public class Generator

{

public void Show<TypeOfValue>(TypeOfValue value)

{

Console.WriteLine(value);

}

}

public class Exercise

{

static int Main()

{

var exo = new Generator();

// Call the version of the function that displays an integer

var Value1 = 246;

exo.Show(Value1);

// Call the version of the function that displays a character

var Value2 = 'G';

exo.Show(Value2);

// Call the version of the function that displays a decimal

var Value3 = 355.65;

exo.Show(Value3);

return 0;

}

}

Como alternativa, puede escribir el nombre del método, seguido de corchetes angulares. Dentro de los corchetes, ingrese el tipo de datos del valor que se procesará. Después de los corchetes angulares, abra los paréntesis y, dentro de ellos, escriba el valor constante que se procesará. He aquí algunos ejemplos:

using System;

public class Generator

{

public void Show<TypeOfValue>(TypeOfValue value)

{

Console.WriteLine(value);

}

}

public class Program

{

static int Main()

{

var exo = new Generator();

// Call the version of the function that displays an integer

var Value1 = 246;

exo.Show<int>(Value1);

// Call the version of the function that displays a character

var Value2 = 'G';

exo.Show<char>(Value2);

// Call the version of the function that displays a decimal

var Value3 = 355.65;

exo.Show<double>(Value3);

return 0;

}

}

También puede declarar el valor como una constante antes de pasarlo al método.

Para especificar el tipo de parámetro de un método genérico al llamar, cambie el método Main() de la siguiente manera:

class Program

{

static void Main()

{

. . .

Console.WriteLine("-=- List of Items -=-");

for (int i = 0; i < exo.Count(); i++)

{

Exercise.CItem One = exo.Retrieve(i);

exo.ShowItem<double>(One.Item);

}

Console.WriteLine("\nNumber of Items: {0}\n", exo.Count());

}

}

**Un método genérico con varios parámetros**

**Introducción**

Al igual que un método puede tomar un argumento, puede tomar varios parámetros genéricos. Puede pasar un argumento como un tipo conocido y el otro como un tipo genérico. Aquí hay un ejemplo:

**Modificar la clase Generator**

using System;

public class Generator

{

public void Show<TypeOfValue>(string msg, TypeOfValue value)

{

Console.WriteLine("{0}: {1}", msg, value);

}

}

public class Exercise

{

static int Main()

{

var exo = new Generator();

exo.Show<int>("Integer", 246);

exo.Show<char>("Character", 'G');

exo.Show<double>("Decimal", 355.65);

return 0;

}

}

resultado

Integer: 246

Character: G

Decimal: 355.65

Press any key to continue . . .

Aunque directamente pasamos los valores al método cuando lo llamamos, primero puede declarar una variable antes de pasarla al método. He aquí algunos ejemplos:

Modificar la clase Generator

using System;

public class Generator

{

public void Show<TypeOfValue>(string msg, TypeOfValue value)

{

Console.WriteLine("{0}: {1}", msg, value);

}

}

public class Exercise

{

static int Main()

{

var exo = new Generator();

var message = "Integer";

const int iValue = 246;

exo.Show<int>(message, iValue);

message = "Character";

const char cValue = 'G';

exo.Show<char>(message, cValue);

message = "Decimal";

const double dValue = 355.65;

exo.Show<double>(message, dValue);

return 0;

}

}

* Acceda al archivo **StoreItem**.cs
* Para crear y utilizar un método con varios parámetros, realice los cambios siguientes:

|  |
| --- |
| namespace CommercialStore1  {  public class StoreItem  {  . . .  public void ShowItem<T>(string content, int index, T item)  {  Console.WriteLine("{0} {1}: {2}", content, index, item);  }  }  } |

Acceda al archivo Program.cs y cámbielo de la siguiente manera:

|  |
| --- |
| namespace CommercialStore1  {  class Program  {  static void Main()  {  . . .  for (int i = 0; i < exo.Count(); i++)  {  Exercise.CItem One = exo.Retrieve(i);  exo.ShowItem<double>("Item", i+1, One.Item);  }  Console.WriteLine("\nNumber of Items: {0}\n", exo.Count());  }  }  } |

Resultado

|  |
| --- |
| -=- List of Items -=-  Item 1: 97.43  Item 2: 274.87  Item 3: 8.7873  Item 4: 2764.4  Item 5: 92.4662  Item 6: 66800.85  Number of Items: 6  Press any key to continue . . . |

**Un método genérico con varios tipos de parámetros**

Como se ha visto anteriormente, puede pasar diferentes argumentos a un método. También puede pasar diferentes tipos de parámetros, en cualquier orden apropiada de su elección, a un método. Para pasar dos tipos de parámetros a un método, dentro de su <> operador, ingrese los nombres de dos tipos de parámetros separados por una coma. Aquí hay un ejemplo:

public class Generator

{

public void Show<FirstType, SecondType>()

{

}

}

Si desea utilizar los tipos de parámetro, puede pasar un argumento para cada uno al método. Recuerde que cada tipo de parámetro representa un tipo de datos; Por lo que puede utilizarlo como el tipo de un argumento. He aquí algunos ejemplos:

public class Generator

{

public void Show<FirstType, SecondType>(FirstType first,

SecondType second)

{

}

}

En el cuerpo del método, puede utilizar los argumentos como mejor le parezca. Por ejemplo, puede mostrar sus valores pasándolos al método Console.WriteLine (). Aquí hay un ejemplo:

public class Generator

{

public void Show<FirstType, SecondType>(FirstType first,

SecondType second)

{

Console.WriteLine("First: {0}\nSecond: {1}\n", first, second);

}

}

**Llamar a un método genérico con varios tipos de parámetros**

Para llamar a un método que tome varios parámetros, simplemente puede pasar el valor de cada argumento. Aquí hay un ejemplo:

using System;

public class Generator

{

public void Show<FirstType, SecondType>(FirstType first,

SecondType second)

{

Console.WriteLine("First: {0}\nSecond: {1}\n", first, second);

}

}

public class Exercise

{

static int Main()

{

var exo = new Generator();

var iValue = 246;

var message = "Some Message";

exo.Show(message, iValue);

return 0;

}

}

Resultado

First: Some Message

Second: 246

Press any key to continue . . .

Una alternativa es especificar el tipo de cada argumento. Para ello, dentro del <> operador en el lado derecho del nombre del método, introduzca los tipos de datos separados por una coma. He aquí algunos ejemplos:

using System;

public class Generator

{

public void Show<FirstType, SecondType>(FirstType first,

SecondType second)

{

Console.WriteLine("First: {0}\nSecond: {1}\n", first, second);

}

}

public class Exercise

{

static int Main()

{

var exo = new Generator();

var iValue = 246;

var message = "Some Message";

exo.Show(message, iValue);

iValue = 85;

var cValue = 'G';

exo.Show<int, char>(iValue, cValue);

var weeklyHours = 42.50d;

var hourlySalary = 25.05;

exo.Show<double, double>(weeklyHours, hourlySalary);

return 0;

}

}

Resultado

First: Some Message

Second: 246

First: 85

Second: G

First: 42.5

Second: 25.05

Observe que los argumentos pueden ser del mismo tipo o tipos diferentes. Depende de usted determinar el tipo de un argumento en particular al llamar al método.

**Clases genéricas**

 Introducción

Como un método, una clase puede ser creada como un genérico. Cuando se crea una clase como genérica, se le pide que procese un valor sin saber qué tipo es ese valor. Esto significa que la clase conocerá el tipo de valor sólo cuando se llama.

Para crear una clase genérica, en el lado derecho del nombre de la clase, escriba el operador <> e ingrese un nombre para el tipo de parámetro. Aquí hay un ejemplo:

public class Exercise<TypeOfValue>

{

}

Este tipo de parámetro es sólo un representante de un tipo de datos. Como tipo de datos, puede utilizarlo para declarar una variable en el cuerpo de la clase. Aquí hay un ejemplo:

public class Exercise<TypeOfValue>

{

public TypeOfValue value;

}

Después de declarar dicha variable, puede usarla en su aplicación. Por ejemplo, puede acceder a él fuera de la clase con el operador de período. Dentro de la clase, una forma de utilizar la variable es mostrar su valor utilizando uno de los métodos de la clase. Aquí hay un ejemplo:

public class Exercise<TypeOfValue>

{

public TypeOfValue value;

public void Show()

{

Console.WriteLine("Value: {0}\n", value);

}

}

**Práctica 05**

Agregar una nuevo Aplicación de consola, nombre: CommercialStore2

Agregar una nueva clase, nombre: ListOfItems

Cambie el archivo como sigue:

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace CommercialStore2  {  public class ListOfItems  {  public class CItem  {  public double Item;  public CItem Next;  }  public ListOfItems()  {  Head = null;  Size = 0;  }  public CItem Head;  public int Size;  public int Count()  {  return Size;  }  public int Add(CItem NewItem)  {  CItem Sample = new CItem();  Sample = NewItem;  Sample.Next = Head;  Head = Sample;  return Size++;  }  public CItem Retrieve(int Position)  {  CItem Current = Head;  for (int i = Count() - 1; i > Position && Current != null; i--)  {  Current = Current.Next;  }  return Current;  }  }  } |

En Program.cs va el siguiente código:

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace CommercialStore2  {  public class Program  {  static void Main()  {  ListOfItems exo = new ListOfItems();  ListOfItems.CItem Part;  Part = new ListOfItems.CItem();  Part.Item = 97.43;  exo.Add(Part);  Part = new ListOfItems.CItem();  Part.Item = 274.87;  exo.Add(Part);  Part = new ListOfItems.CItem();  Part.Item = 8.7873;  exo.Add(Part);  Part = new ListOfItems.CItem();  Part.Item = 2764.4;  exo.Add(Part);  Part = new ListOfItems.CItem();  Part.Item = 92.4662;  exo.Add(Part);  Part = new ListOfItems.CItem();  Part.Item = 66800.85;  exo.Add(Part);  Console.WriteLine("-=- List of Items -=-");  for (int i = 0; i < exo.Count(); i++)  {  ListOfItems.CItem One = exo.Retrieve(i);  Console.WriteLine("Item: {0}", One.Item);  }  Console.WriteLine("\nNumber of Items: {0}\n", exo.Count());  }  }  } |

Resultado

|  |
| --- |
| -=- List of Items -=-  Item: 97.43  Item: 274.87  Item: 8.7873  Item: 2764.4  Item: 92.4662  Item: 66800.85  Number of Items: 6  Press any key to continue . . . |

* Definir Generic para el código presentado

using System;

public class Generator<TypeOfValue>

{

public TypeOfValue value;

public void Show()

{

Console.WriteLine("Value: {0}\n", value);

}

}

public class Exercise

{

static int Main()

{

Generator<int> exo = new Generator<int>();

return 0;

}

}

También puede declarar una variable de un tipo genérico utilizando la palabra clave var. Para ello, utilice var en el lado izquierdo del nombre de la variable, omita el operador <> y el tipo de datos. Esto se haría de la siguiente manera:

public class Exercise

{

static int Main()

{

var exo = new Generator<int>();

return 0;

}

}

Después de declarar la variable, puede acceder a los miembros de la clase utilizando el operador período. He aquí algunos ejemplos:

using System;

public class Generator<TypeOfValue>

{

public TypeOfValue value;

public void Show()

{

Console.WriteLine("Value: {0}\n", value);

}

}

public class Exercise

{

static int Main()

{

var exo = new Generator<int>();

var iValue = 246;

exo.value = iValue;

exo.Show();

return 0;

}

}

**Pasar un tipo de parámetro a un método**

Vimos que se podía declarar una variable de un tipo de parámetro en la clase genérica. Otra forma de utilizarlo es pasarlo como un argumento a un método y convertirlo en un tipo de parámetro. Como se ha visto anteriormente, puede utilizar el argumento como mejor le parezca. Por ejemplo, puede mostrar su valor a la consola. Aquí hay un ejemplo:

using System;

public class Generator<TypeOfValue>

{

public void Show(TypeOfValue value)

{

Console.WriteLine("Value: {0}\n", value);

}

}

public class Exercise

{

static int Main()

{

var exo = new Generator<int>();

var iValue = 246;

exo.Show(iValue);

return 0;

}

}

De la misma manera, puede pasar el tipo de parámetro a un constructor de la clase. Aquí hay un ejemplo:

public class Generator<TypeOfValue>

{

private TypeOfValue val;

public Exercise(TypeOfValue v)

{

val = v;

}

}

**Devolución de un tipo de parámetro**

Además, o en lugar de pasar un tipo de parámetro, puede crear un método que devuelva un tipo de parámetro. Una vez más, puede observar principalmente las reglas que revisamos para devolver un valor de un método. Aquí hay un ejemplo:

using System;

public class Generator<TypeOfValue>

{

private TypeOfValue val;

public Generator(TypeOfValue v)

{

val = v;

}

public TypeOfValue GetValue()

{

return val;

}

}

public class Exercise

{

static int Main()

{

var exo = new Generator<double>(35.65);

Console.WriteLine("Value: {0}\n", exo.GetValue());

return 0;

}

}

**Devolución de un tipo de parámetro**

Abra el archivo de origen ListOfItems.cs

Para aplicar lo que hemos revisado, cambie el archivo de la siguiente manera:

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace CommercialStore2  {  public class ListOfItems  {  public class CItem<T>  {  public double Item;  public CItem<T> Next;  }  public ListOfItems()  {  Head = null;  Size = 0;  }  public CItem<double> Head;  public int Size;  public int Count()  {  return Size;  }  public int Add(CItem<double> NewItem)  {  CItem<double> Sample = new CItem<double>();  Sample = NewItem;  Sample.Next = Head;  Head = Sample;  return Size++;  }  public CItem<double> Retrieve(int Position)  {  CItem<double> Current = Head;  for (int i = Count() - 1; i > Position && Current != null; i--)  {  Current = Current.Next;  }  return Current;  }  }  } |

En Program.cs realizar los siguientes:

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace CommercialStore2  {  public class Program  {  static void Main()  {  ListOfItems exo = new ListOfItems();  ListOfItems.CItem<double> Part;  Part = new ListOfItems.CItem<double>();  Part.Item = 97.43;  exo.Add(Part);  Part = new ListOfItems.CItem<double>();  Part.Item = 274.87;  exo.Add(Part);  Part = new ListOfItems.CItem<double>();  Part.Item = 8.7873;  exo.Add(Part);  Part = new ListOfItems.CItem<double>();  Part.Item = 2764.4;  exo.Add(Part);  Part = new ListOfItems.CItem<double>();  Part.Item = 92.4662;  exo.Add(Part);  Part = new ListOfItems.CItem<double>();  Part.Item = 66800.85;  exo.Add(Part);  Console.WriteLine("-=- List of Items -=-");  for (int i = 0; i < exo.Count(); i++)  {  ListOfItems.CItem<double> One = exo.Retrieve(i);  Console.WriteLine("Item: {0}", One.Item);  }  Console.WriteLine("\nNumber of Items: {0}\n", exo.Count());  }  }  } |

Probar

**Una propiedad del tipo de parámetro**

Puede crear una propiedad que sea del tipo de parámetro de la clase genérica. No hay ninguna regla significativa a seguir al crear la propiedad, excepto que debe recordar que, en el momento de crear la propiedad, la clase no conoce el tipo del parámetro. Aquí hay un ejemplo:

using System;

public class Generator<TypeOfValue>

{

private TypeOfValue val;

public TypeOfValue Value

{

get { return val; }

set { val = value; }

}

public void Show()

{

Console.WriteLine("Value: {0}\n", val);

}

}

public class Exercise

{

static int Main()

{

var exo = new Generator<int>();

exo.Value = 246;

exo.Show();

return 0;

}

}

**Una clase genérica con múltiples parámetros**

Uso de varios parámetros de tipo

Como se hace para los métodos genéricos, al crear una clase genérica, puede especificar más de un tipo de parámetro. Para ello, en el operador <>, después del primer tipo genérico, introduzca una coma y otro tipo genérico. Aquí hay un ejemplo:

public class Generator<T, V>

{

}

Si sabe que los parámetros serán del mismo tipo, puede usar un método para procesar ambos. De lo contrario, puede declarar los miembros necesarios para cada tipo. También puede crear un método que tome muchos argumentos con cada argumento de un tipo particular. He aquí algunos ejemplos:

using System;

public class Generator<T, V>

{

private T t;

private V v;

public void SetTValue(T value)

{

t = value;

}

public T GetTValue()

{

return t;

}

public void SetVValue(V value)

{

v = value;

}

public V GetVValue()

{

return v;

}

public void Show(T tValue, V vValue)

{

Console.WriteLine("First: {0}\nSecond: {1}", tValue, vValue);

}

}

Al declarar una variable para la clase, asegúrese de especificar apropiadamente la lista de tipos de parámetros. He aquí dos ejemplos:

public class Exercise

{

static int Main()

{

Generator<int, int> IntTypes = new Generator<int, int>();

IntTypes.SetTValue(246);

IntTypes.SetVValue(6088);

IntTypes.Show(IntTypes.GetTValue(), IntTypes.GetVValue());

Generator<double, double> DoubleTypes = new Generator<double, double>();

DoubleTypes.SetTValue(355.65);

DoubleTypes.SetVValue(1785.426);

DoubleTypes.Show(DoubleTypes.GetTValue(), DoubleTypes.GetVValue());

Generator<short, decimal> Disparate = new Generator<short, decimal>();

DoubleTypes.SetTValue(42);

DoubleTypes.SetVValue(245580.35);

DoubleTypes.Show(DoubleTypes.GetTValue(), DoubleTypes.GetVValue());

return 0;

}

}

También puede declarar la variable utilizando la palabra clave var. Para ello, en el lado izquierdo del operador de asignación, escriba sólo la palabra clave var y el nombre del operador. Aquí hay un ejemplo:

**public class Exercise**

**{**

**static int Main()**

**{**

**var IntTypes = new Generator<int, int>();**

**IntTypes.SetTValue(246);**

**IntTypes.SetVValue(6088);**

**IntTypes.Show(IntTypes.GetTValue(), IntTypes.GetVValue());**

**var DoubleTypes = new Generator<double, double>();**

**DoubleTypes.SetTValue(355.65);**

**DoubleTypes.SetVValue(1785.426);**

**DoubleTypes.Show(DoubleTypes.GetTValue(), DoubleTypes.GetVValue());**

**var Disparate = new Generator<short, decimal>();**

**DoubleTypes.SetTValue(42);**

**DoubleTypes.SetVValue(245580.35);**

**DoubleTypes.Show(DoubleTypes.GetTValue(), DoubleTypes.GetVValue());**

**return 0;**

**}**

**}**

**Resultado**

First: 246

Second: 6088

First: 355.65

Second: 1785.426

First: 42

Second: 245580.35

Press any key to continue . . .

Si una clase genérica tiene más de un tipo de parámetro, no tiene que ser del mismo tipo. En el momento en que está creando la clase, no puede especificar sus tipos, pero se puede anticipar que serían diferentes. Es cuando usted declara la variable que usted necesitaría para determinar sus tipos exactos.

Incluso si los parámetros son de tipos primitivos, primero puede declarar las variables y pasarlas a la clase.

**Una clase como un tipo de parámetro**

Hasta ahora, en nuestros ejemplos, tratamos el tipo de parámetro como un tipo de datos primitivo. Un tipo de parámetro también puede ser una clase formal, ya sea uno que usted mismo creó o uno que existe como parte del lenguaje C #. Al crear la clase genérica, debe seguir todas las reglas que hemos revisado hasta ahora para classess genérico. Aquí está una clase tan simple:

public class Generator

{

public void Show(TypeOfValue val)

{

Console.WriteLine("{0}\n", val.ToString());

}

}

Como se mencionó anteriormente, la clase que sería procesada por el genérico debe haber sido previamente creada para que pueda usarse como un parámetro. Al declarar una variable de la clase genérica, asegúrese de introducir el nombre de la clase normal en lugar del tipo de parámetro. Todo lo demás es como hemos hecho hasta ahora. Aquí hay un ejemplo:

using System;

public class FourSideGeometricFigure

{

private string nm;

private double bs;

private double hg;

public string Name

{

get { return nm; }

set { nm = value; }

}

public double Base

{

get { return bs; }

set { bs = value; }

}

public double Height

{

get { return hg; }

set { hg = value; }

}

public double Area

{

get { return bs \* hg; }

}

public override string ToString()

{

string result = "Type: " + nm + "\n" +

"Base: " + bs.ToString() + "\n" +

"Height: " + hg.ToString() + "\n" +

"Area: " + Area.ToString();

return result;

}

}

public class Generator<TypeOfValue>

{

public void Show(TypeOfValue val)

{

Console.WriteLine("{0}\n", val.ToString());

}

}

public class Exercise

{

static int Main()

{

var sqr = new FourSideGeometricFigure();

sqr.Name = "Square";

sqr.Base = 36.82;

sqr.Height = 36.82;

Generator<FourSideGeometricFigure> exoSquare =

new Generator<FourSideGeometricFigure>();

exoSquare.Show(sqr);

FourSideGeometricFigure rect = new FourSideGeometricFigure();

rect.Name = "Rectangle";

rect.Base = 52.94;

rect.Height = 27.58;

Generator<FourSideGeometricFigure> exoRect =

new Generator<FourSideGeometricFigure>();

exoRect.Show(rect);

return 0;

}

}

También puede declarar la variable utilizando la palabra clave var. He aquí algunos ejemplos:

public class Exercise

{

static int Main()

{

var sqr = new FourSideGeometricFigure();

sqr.Name = "Square";

sqr.Base = 36.82;

sqr.Height = 36.82;

var exoSquare = new Generator<FourSideGeometricFigure>();

exoSquare.Show(sqr);

FourSideGeometricFigure rect = new FourSideGeometricFigure();

rect.Name = "Rectangle";

rect.Base = 52.94;

rect.Height = 27.58;

var exoRect = new Generator<FourSideGeometricFigure>();

exoRect.Show(rect);

return 0;

}

}

Resultado

Type: Square

Base: 36.82

Height: 36.82

Area: 1355.7124

Type: Rectangle

Base: 52.94

Height: 27.58

Area: 1460.0852

Press any key to continue . . .

Clases genéricas y herencia

Consideremos las siguientes figuras geométricas:

|  |  |  |  |
| --- | --- | --- | --- |
| Square | http://www.functionx.com/illustrations/rectangle5.gif | http://www.functionx.com/illustrations/trapezoid1.gif | Parallelogram |
| **Square** | **Rectangle** | **Trapezoid** | **Parallelogram** |

Observe que éstas son figuras geométricas con cada uno que tiene cuatro lados. De lo que sabemos hasta ahora, podemos crear una clase base para prepararla para la herencia. Si la clase es muy general, podemos hacerla genérica. Podemos configurar un tipo de datos como un tipo desconocido, anticipando que las dimensiones de la figura se pueden considerar como tipos enteros o de doble precisión. Aquí hay un ejemplo:

using System;

public class Quadrilateral<T>

{

protected T \_base;

protected T \_height;

protected string \_name;

public virtual T Base

{

get { return \_base; }

set { \_base = value; }

}

public virtual T Height

{

get { return \_height; }

set { \_height = value; }

}

public virtual string Name

{

get { return \_name; }

set { \_name = value; }

}

public Quadrilateral()

{

\_name = "Quadrilateral";

}

public Quadrilateral(string name)

{

\_name = "Quadrilateral";

}

public Quadrilateral(T bs, T height)

{

\_name = "Quadrilateral";

\_base = bs;

\_height = height;

}

public Quadrilateral(string name, T bs, T height)

{

\_name = name;

\_base = bs;

\_height = height;

}

public virtual string Describe()

{

return "A quadrilateral is a geometric figure with four sides";

}

public virtual void ShowCharacteristics()

{

Console.WriteLine("Geometric Figure: {0}", Name);

Console.WriteLine("Description: {0}", Describe());

Console.WriteLine("Base: {0}", Base);

Console.WriteLine("Height: {0}", Height);

}

}

public class Exercise

{

static int Main()

{

// Trapezoid with equal sides

var Kite = new Quadrilateral<double>("Beach Kite", 18.64, 18.64);

Kite.ShowCharacteristics();

Console.WriteLine();

// Rectangle, in meters

var BasketballStadium = new Quadrilateral<Byte>();

BasketballStadium.Name = "Basketball Stadium";

BasketballStadium.Base = 15;

BasketballStadium.Height = 28;

BasketballStadium.ShowCharacteristics();

Console.WriteLine();

return 0;

}

}

Resultado

Geometric Figure: Beach Kite

Description: A quadrilateral is a geometric figure with four sides

Base: 18.64

Height: 18.64

Geometric Figure: Basketball Stadium

Description: A quadrilateral is a geometric figure with four sides

Base: 15

Height: 28

Press any key to continue . . .

Si usted tiene una clase genérica que puede servir como base para otra clase, puede derivar una clase de la genérica. Para ello, utilice la fórmula que aplicamos al derivar una clase, pero siga el nombre de cada clase con <>. Dentro del operador <>, ingrese el mismo identificador para indicar que la clase es un tipo genérico que se basa en otra clase genérica. Aquí hay un ejemplo:

public class Square<T> : Quadrilateral<T>

{

}

En el cuerpo de la nueva clase, puede utilizar el tipo de parámetro como mejor le parezca. Por ejemplo, puede declarar algunas variables miembro de ese tipo. Puede crear métodos que devuelvan el tipo de parámetro o puede pasar argumentos del tipo de parámetro. Al implementar los métodos de la nueva clase, utilice las variables miembro del parámetro y los argumentos basados en el tipo de parámetro como mejor le parezca. A continuación, puede declarar una variable de la clase y utilizarla como lo hemos hecho hasta ahora para otras clases genéricas. Aquí hay un ejemplo:

using System;

public class Quadrilateral<T>

{

protected T \_base;

protected T \_height;

protected string \_name;

public virtual T Base

{

get { return \_base; }

set { \_base = value; }

}

public virtual T Height

{

get { return \_height; }

set { \_height = value; }

}

public virtual string Name

{

get { return \_name; }

set { \_name = value; }

}

public Quadrilateral()

{

\_name = "Quadrilateral";

}

public Quadrilateral(string name)

{

\_name = "Quadrilateral";

}

public Quadrilateral(T bs, T height)

{

\_name = "Quadrilateral";

\_base = bs;

\_height = height;

}

public Quadrilateral(string name, T bs, T height)

{

\_name = name;

\_base = bs;

\_height = height;

}

public virtual string Describe()

{

return "A quadrilateral is a geometric figure with four sides";

}

public virtual void ShowCharacteristics()

{

Console.WriteLine("Geometric Figure: {0}", Name);

Console.WriteLine("Description: {0}", Describe());

Console.WriteLine("Base: {0}", Base);

Console.WriteLine("Height: {0}", Height);

}

}

public class Square<T> : Quadrilateral<T>

{

public Square()

{

\_name = "Square";

}

public Square(string name)

{

\_name = "Square";

}

public Square(T side)

{

\_name = "Square";

\_base = side;

\_height = side;

}

public Square(string name, T side)

{

\_name = name;

\_base = side;

\_height = side;

}

public override string Describe()

{

return "A square is a quadrilateral with four equal sides";

}

public override void ShowCharacteristics()

{

Console.WriteLine("Geometric Figure: {0}", Name);

Console.WriteLine("Description: {0}", Describe());

Console.WriteLine(" {0}", Describe());

Console.WriteLine("Side: {0}", Base);

}

}

public class Exercise

{

static int Main()

{

// Rectangle, in meters

var plate = new Square<Byte>();

plate.Name = "Plate";

plate.Base = 15;

plate.Height = 28;

plate.ShowCharacteristics();

Console.WriteLine();

return 0;

}

}

This would produce:

Geometric Figure: Plate

Description: A quadrilateral is a geometric figure with four sides

A square is a quadrilateral with four equal sides

Side: 15

Press any key to continue . . .

Clases Generales e Interfaces

Como se hace para una clase genérica, puede crear una interfaz genérica que sirva como base para clases genéricas. Para proceder, al crear la interfaz, siga su nombre con una declaración <> y, dentro del operador <>, ingrese un identificador para el tipo de parámetro. Aquí hay un ejemplo:

public interface IGeometry<T>

{

string Name { get; set; }

void Display();

}

Dado que se trata de una interfaz genérica, como una clase de interfaz, al derivar una clase de ella, siga la fórmula que revisamos para heredar una clase genérica. Aquí hay un ejemplo:

using System;

public interface IGeometry<T>

{

string Name { get; set; }

void Display();

}

public class Round<T> : IGeometry<T>

{

}

Al implementar la clase derivada, debe observar todas las reglas que se aplican a la derivación de interfaz. Aquí hay un ejemplo:

using System;

public interface IGeometry<T>

{

string Name { get; set; }

void Display();

}

public class Round<T> : IGeometry<T>

{

private string \_name;

public Round()

{

\_name = "Unknown";

}

public Round(string name)

{

\_name = name;

}

public virtual string Name

{

get { return \_name; }

set { \_name = value; }

}

public virtual void Display()

{

Console.WriteLine("Name: {0}", Name);

}

}

public class Exercise

{

static int Main()

{

var rnd = new Round<decimal>();

rnd.Name = "General Round Shape";

rnd.Display();

Console.WriteLine();

return 0;

}

}

Resultado

Name: General Round Shape

Press any key to continue . . .

De la misma manera, puede derivar una clase genérica de otra clase genérica que se deriva de una interfaz genérica.

Limitar una clase genérica

Imagine que cree una interfaz regular como la siguiente:

Entonces imagine que usted derive una clase regular de ella. Aquí hay un ejemplo:

using System;

public interface IGeometry

{

string Name { get; set; }

void Display();

}

public class Round : IGeometry

{

private string \_name;

private double \_rad;

public Round()

{

\_name = "Unknown";

}

public Round(string name)

{

\_name = name;

\_rad = 0.00;

}

public Round(string name, double radius)

{

\_name = name;

\_rad = radius;

}

public virtual string Name

{

get { return \_name; }

set { \_name = value; }

}

public double Radius

{

get { return \_rad; }

set

{

\_rad = (value <= 0) ? 0.00 : value;

}

}

public virtual void Display()

{

Console.WriteLine("Name: {0}", Name);

Console.WriteLine("Radius: {0}", Radius);

}

}

public class Exercise

{

static int Main()

{

var rnd = new Round();

rnd.Name = "General Round Shape";

rnd.Radius = 38.24;

rnd.Display();

Console.WriteLine();

return 0;

}

}

Usted puede ser tentado a derivar cualquier tipo de clase de ella. Una de las características de los genéricos es que puede crear una clase que debe implementar la funcionalidad de una determinada clase abstracta de su elección. Por ejemplo, al crear una clase genérica, puede obligarla a implementar la funcionalidad de una determinada interfaz o puede asegurarse de que la clase se deriva de una clase base específica. Esto asegurará que la clase genérica seguramente contiene alguna funcionalidad útil.

Para crear una restricción en una clase genérica, después del operador <TypeName>, escriba whereTypeName: seguido por la regla que debe seguir la clase. Por ejemplo, es posible que desee que la clase genérica implemente la funcionalidad de una clase predefinida. Puede crear la clase genérica de la siguiente manera:public interface IGeometry

{

}

public class Round : IGeometry

{

}

public class Sphere<T>

where T : Round

{

}

Después de crear la clase, debe implementar los miembros virtuales de whereclass / interface, utilizando las reglas de las clases genéricas, como hemos hecho hasta ahora. Al declarar una variable para la clase genérica, en su operador <>, debe introducir un objeto de la clase base. Aquí hay un ejemplo:using System;

public interface IGeometry

{

string Name { get; set; }

void Display();

}

public class Round : IGeometry

{

private string \_name;

private double \_rad;

public Round()

{

\_name = "Unknown";

}

public Round(string name)

{

\_name = name;

\_rad = 0.00;

}

public Round(string name, double radius)

{

\_name = name;

\_rad = radius;

}

public virtual string Name

{

get { return \_name; }

set { \_name = value; }

}

public double Radius

{

get { return \_rad; }

set

{

\_rad = (value <= 0) ? 0.00 : value;

}

}

public virtual void Display()

{

Console.WriteLine("Name: {0}", Name);

Console.WriteLine("Radius: {0}", Radius);

}

}

public class Sphere<T>

where T : Round

{

private T \_t;

public Sphere() { }

public Sphere(T fig)

{

\_t = fig;

}

public T Figure

{

get { return \_t; }

set { \_t = value; }

}

}

public class Exercise

{

static int Main()

{

var rnd = new Round();

rnd.Name = "Circle";

rnd.Radius = 60.12;

Sphere<Round> sph = new Sphere<Round>();

sph.Figure = rnd;

Console.WriteLine("Circle Characteristics");

Console.WriteLine("Name: {0}", sph.Figure.Name);

Console.WriteLine("Radius: {0}", sph.Figure.Radius);

Console.WriteLine();

return 0;

}

}

Resultado

Circle Characteristics

Name: Circle

Radius: 60.12

Press any key to continue . . .

**Práctica 4**

Una aplicación .NET típica tiene algún tipo de archivo de configuración de la aplicación, la mayoría de las veces un web.config o app.config dependiendo del tipo de proyecto. El archivo de configuración basado en XML tiene una sección llamada appSettings donde puede agregar sus configuraciones personalizadas como pares clave-valor.

1. En proyecto de consola: GenericConsole agregar en el app.config los appSettings

<configuration>

    <appSettings>

        <add key="environment" value="alpha"/>

        <add key="max-loops" value="10"/>

        <add key="cheat-mode-on" value="true"/>

    </appSettings>

</configuration>

**Nota**: en el nuevo núcleo de ASP.NET, el archivo de configuración basado en XML tradicional se reemplaza por un archivo JSON.

Tenga en cuenta que todos los valores se almacenan como cadenas en el archivo XML por supuesto. Sin embargo, queremos convertirlos en sus tipos reales: string, int y boolean respectivamente. Hay varias maneras de resolver el problema, pero creo que los genéricos proporcionan una buena opción.

1. Leer una sola configuración del archivo de configuración de la siguiente manera e imprimir por consola sus valor.

string key = "environment";

string value = ConfigurationManager.AppSettings[key];

La clase **ConfigurationManager** se encuentra en el espacio de nombres **System.Configuration**. Sólo estará disponible si agrega una referencia a la DLL de **System.Configuration** en su proyecto

1. Agregar referencia a **System.Configuration**
   1. Agregar una clase ConfigurationReaderService

public class ConfigurationReaderService

{

    public string ReadConfigurationValue(string key)

    {

        return ConfigurationManager.AppSettings[key];

    }

}

1. Probar ConfigurationReaderService y leer el ajuste de max-loops:

ConfigurationReaderService configService = new ConfigurationReaderService();

string key = "max-loops";

string maxLoopsRaw = configService.ReadConfigurationValue(key);

if (!string.IsNullOrEmpty(maxLoopsRaw))

{

    int maxLoops;

    if (int.TryParse(maxLoopsRaw, out maxLoops))

    {

        Debug.WriteLine(string.Format("Converted: {0}", maxLoops));

    }

    else

    {

        Debug.WriteLine(string.Format("Could not convert: {0}", maxLoopsRaw));

    }

}

else

{

    Debug.WriteLine(string.Format("There's no such setting: {0}", key));

}

El código escribe "Converted: 10" en la ventana de depuración.

Si cambiamos el valor de "max-loops" a una cadena que no se puede convertir en un int, como "ElvisPresley", entonces obtendremos "Could not convert: ElvisPresley". Si eliminamos o cambiamos el nombre de la configuración de "max-loops" del archivo de configuración, obtendremos "No hay tal configuración: max-loops".  
Eso podría ser todo lo que necesita: llamar a ese servicio simple y tratar con la variable de cadena después. Sin embargo, creo que un servicio de lectura de configuración debe ser un poco más versátil. Lo ideal sería poder realizar la conversión de tipo por su cuenta para que el llamador pueda centrarse en el valor convertido directamente. Esto es para que el llamador puede evitar tener que pasar por todos los pasos de validación.  
Crear una solución con generic **ConfigurationReaderService y luego realizar la prueba.**

Práctica 5