BridgeLabz

Employability Delivered

.NET Core Concepts – Generics

.Net Core Concepts

BridgeLabz
Employability Delivered

- .Net Exceptions
- .Net Reflections
- .Net Annotations
- *.Net Generics*
- *.Net Properties*
- .Net OpenCSV
- .Net JSON using Gson

.NET Generics

BridgeLabz
Employability Delivered

.NET Generics enables programmers to support Programming Logic and Structure for multiple data types. They can be applied to a Method or a Class –

- **Generic Methods –** Enables programmers to specify with a single method declaration, to accommodate a set of related methods.
- **Generic Class –** Enables programmers to specify with a single class declaration, to accommodate a set of related types.

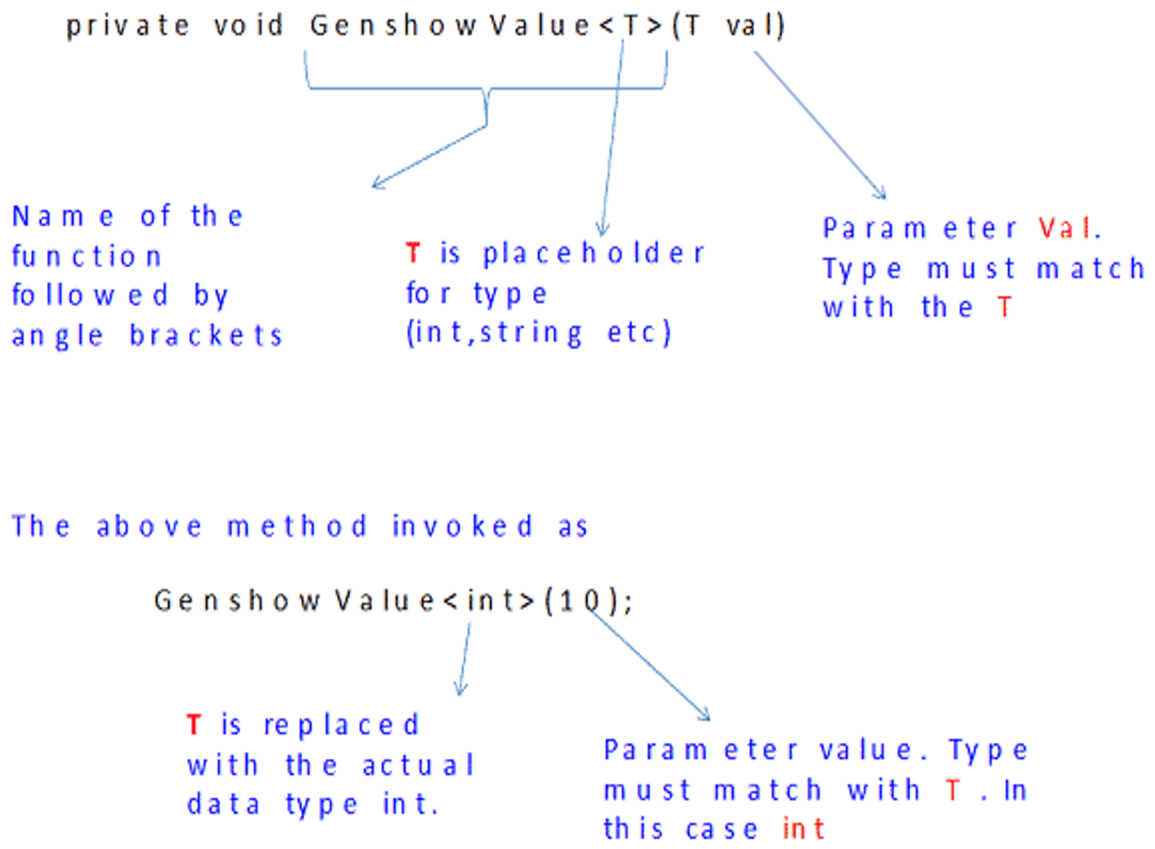
.NET Generic Methods

BridgeLabz
Employability Delivered

- Generic method can be declared to handle arguments of different types.
- Based on the types of the arguments passed to the generic method, the compiler handles each method call appropriately.

BridgeLabz
Employability Delivered

.NET Generic Method Declaration



UC 1

Given an array of Integer, Double and Character, write a program to print the same

- Create PrintArray class and define toPrint method to print corresponding elements to console

PrintArray Sample Code

```
using System;

namespace Generics
{
    3 references
    class Program
    {
        1 reference
        public static void toPrint(int[] inputArray)
        {
            foreach(int element in inputArray)
            {
                Console.WriteLine(element);
            }
            Console.WriteLine("--------------------");
        }

        1 reference
        public static void toPrint(double[] inputArray)
        {
            foreach (double element in inputArray)
            {
                Console.WriteLine(element);
            }
            Console.WriteLine("--------------------");
        }

        1 reference
        public static void toPrint(char[] inputArray)
        {
            foreach (char element in inputArray)
            {
                Console.WriteLine(element);
            }
            Console.WriteLine("--------------------");
        }

        0 references
        static void Main(string[] args)
        {
            int[] intArray = { 1, 2, 3, 4, 5 };
            double[] doubleArray = { 1.1, 2.2, 3.3, 4.4 };
            char[] charArray = { 'H', 'E', 'L', 'L', 'O' };

            Program.toPrint(intArray);
            Program.toPrint(doubleArray);
            Program.toPrint(charArray);
        }
    }
}
```

BridgeLabz
Employability Delivered

UC 2

Given an array of Integer, Double and Character, write a program to print the same using Generics

PrintArray Sample Code

```
using System;
using System.Collections.Generic;

namespace Generics
{
    3 references
    class Program
    {
        3 references
        public static void toPrint<T>(T[] inputArray)
        {
            foreach (var element in inputArray)
            {
                Console.WriteLine(element);
            }
            Console.WriteLine("--------------------");
        }
        0 references
        static void Main(string[] args)
        {
            int[] intArray = { 1, 2, 3, 4, 5 };
            double[] doubleArray = { 1.1, 2.2, 3.3, 4.4 };
            char[] charArray = { 'H', 'E', 'L', 'L', 'O' };

            Program.toPrint<int>(intArray);
            Program.toPrint<double>(doubleArray);
            Program.toPrint<char>(charArray);
        }
    }
}
```

BridgeLabz
Employability Delivered

.NET Generic Class

BridgeLabz
Employability Delivered

Generic Class can be declared to handle instance variables of different types. To achieve this the class name is followed by a type parameter section.

As with generic methods, the type parameter section of a generic class can have one or more type parameters separated by commas.

These classes are known as parameterized classes or parameterized types because they accept one or more parameters.

.NET Generic Class Declaration

Example: Generic Class (Simplified)

```
public class ArrayList<E> {

   public E get(int index) { ... }

   ...
}
```

In rest of class, E refers to a type

This says that get returns an E. So, if you created ArrayList<Employee>, get returns an Employee. No typecast required.

This is a highly simplified version of the real java.util.ArrayList class.
That class implements multiple interfaces, and the generic support comes from the interfaces.

PrintArray Sample Code

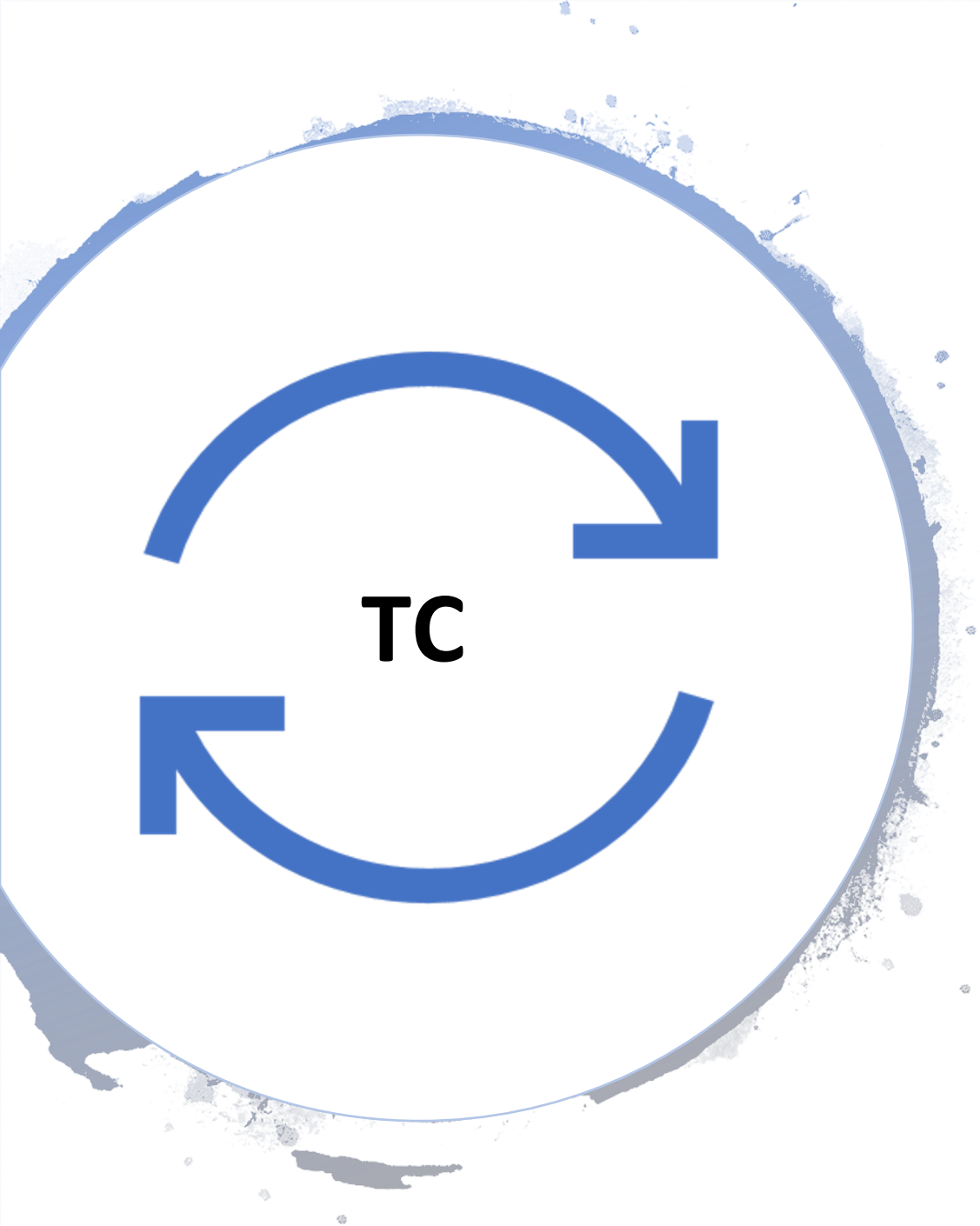
```
namespace Generics
{
    4 references
    public class PrintArray<T>
    {
        private T[] inputArray;
        3 references
        public PrintArray(T[] inputArray)
        {
            this.inputArray = inputArray;
        }

        3 references
        public void toPrint()
        {
           // Console.WriteLine(this.inputArray);
            foreach (var element in inputArray)
            {
                Console.WriteLine(element);
            }
            Console.WriteLine("--------------------");
        }
    }
}

0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        int[] intArray = { 1, 2, 3, 4, 5 };
        double[] doubleArray = { 1.1, 2.2, 3.3, 4.4 };
        char[] charArray = { 'H', 'E', 'L', 'L', 'O' };

        new PrintArray<int>(intArray).toPrint();
        new PrintArray<double>(doubleArray).toPrint();
        new PrintArray<char>(charArray).toPrint();
    }
}
```

BridgeLabz
Employability Delivered



In the following Use Cases Make sure Test Cases

Test your code is working or not with Test Cases

UC 1

Given 3 Integers find the maximum

- Ensure to test code with the Test Case

```
public static int MaximumIntegerNumber(int firstValue, int secondValue, int thirdValue)
{
    if (firstValue.CompareTo(secondValue) > 0 && firstValue.CompareTo(thirdValue) > 0 ||
        firstValue.CompareTo(secondValue) >= 0 && firstValue.CompareTo(thirdValue) > 0 ||
        firstValue.CompareTo(secondValue) > 0 && firstValue.CompareTo(thirdValue) >= 0)
    {
        return firstValue;
    }
    if (secondValue.CompareTo(firstValue) > 0 && secondValue.CompareTo(thirdValue) > 0 ||
        secondValue.CompareTo(firstValue) >= 0 && secondValue.CompareTo(thirdValue) > 0 ||
        secondValue.CompareTo(firstValue) > 0 && secondValue.CompareTo(thirdValue) >= 0)
    {
        return secondValue;
    }
    if (thirdValue.CompareTo(firstValue) > 0 && thirdValue.CompareTo(secondValue) > 0 ||
        thirdValue.CompareTo(firstValue) >= 0 && thirdValue.CompareTo(secondValue) > 0 ||
        thirdValue.CompareTo(firstValue) > 0 && thirdValue.CompareTo(secondValue) >= 0)
    {
        return thirdValue;
    }
    return firstValue;
}
```

UC 2

Given 3 Floats find the maximum

- Ensure to test code with the Test Case

BridgeLabz
Employability Delivered

```
public static double MaximumFloatNumber(double firstValue, double secondValue, double thirdValue)
{
    if (firstValue.CompareTo(secondValue) > 0 && firstValue.CompareTo(thirdValue) > 0 ||
       firstValue.CompareTo(secondValue) >= 0 && firstValue.CompareTo(thirdValue) > 0 ||
       firstValue.CompareTo(secondValue) > 0 && firstValue.CompareTo(thirdValue) >= 0)
    {
        return firstString;
    }
    if (secondValue.CompareTo(firstValue) > 0 && secondValue.CompareTo(thirdValue) > 0 ||
        secondValue.CompareTo(firstValue) >= 0 && secondValue.CompareTo(thirdValue) > 0 ||
        secondValue.CompareTo(firstValue) > 0 && secondValue.CompareTo(thirdValue) >= 0)
    {
        return secondString;
    }
    if (thirdValue.CompareTo(firstValue) > 0 && thirdValue.CompareTo(secondValue) > 0 ||
        thirdValue.CompareTo(firstValue) >= 0 && thirdValue.CompareTo(secondValue) > 0 ||
        thirdValue.CompareTo(firstValue) > 0 && thirdValue.CompareTo(secondValue) >= 0)
    {
        return thirdValue;
    }
    return firstValue;
}
```

UC 3

Given 3 Strings find the maximum

- Ensure to test code with the Test Case

```
public static string MaximumString(string firstString, string secondString, string thirdString)
{
    if (firstString.CompareTo(secondString) > 0 && firstString.CompareTo(thirdString) > 0 ||
        firstString.CompareTo(secondString) >= 0 && firstString.CompareTo(thirdString) > 0 ||
        firstString.CompareTo(secondString) > 0 && firstString.CompareTo(thirdString) >= 0)
    {
        return firstString;
    }
    if (secondString.CompareTo(firstString) > 0 && secondString.CompareTo(thirdString) > 0 ||
        secondString.CompareTo(firstString) >= 0 && secondString.CompareTo(thirdString) > 0 ||
        secondString.CompareTo(firstString) > 0 && secondString.CompareTo(thirdString) >= 0)
    {
        return secondString;
    }
    if (thirdString.CompareTo(firstString) > 0 && thirdString.CompareTo(secondString) > 0 ||
        thirdString.CompareTo(firstString) >= 0 && thirdString.CompareTo(secondString) > 0 ||
        thirdString.CompareTo(firstString) > 0 && thirdString.CompareTo(secondString) >= 0)
    {
        return thirdString;
    }
    return firstString;
}
```

Refactor 1

Refactor all the 3 to One Generic Method and find the maximum

- Ensure the Generic Type extends Comparable
- Make the test case work

Refactor 2

Refactor to create Generic Class to take in 3 variables of Generic Type

- Ensure the Generic Type extends Comparable
- Write parameter constructor
- Write testMaximum method to internally call the static testMaximum method passing the 3 instance variables
- Define new test case to use the Generic Class

```
public class GenericMaximum<T> where T : IComparable
{
    public T firstValue, secondValue, thirdValue;

    12 references | 11/11 passing
    public GenericMaximum(T firstValue, T secondValue, T thirdValue)
    {
        this.firstValue= firstValue;
        this.secondValue = secondValue;
        this.thirdValue = thirdValue;
    }

    1 reference
    public static T MaxValue(T firstValue, T secondValue, T thirdValue)
    {
        if (firstValue.CompareTo(secondValue) > 0 && firstValue.CompareTo(thirdValue) > 0 ||
            firstValue.CompareTo(secondValue) >= 0 && firstValue.CompareTo(thirdValue) > 0 ||
            firstValue.CompareTo(secondValue) > 0 && firstValue.CompareTo(thirdValue) >= 0)
        {
            return firstValue;
        }
        if (secondValue.CompareTo(firstValue) > 0 && secondValue.CompareTo(thirdValue) > 0 ||
            secondValue.CompareTo(firstValue) >= 0 && secondValue.CompareTo(thirdValue) > 0 ||
            secondValue.CompareTo(firstValue) > 0 && secondValue.CompareTo(thirdValue) >= 0)
        {
            return secondValue;
        }
        if (thirdValue.CompareTo(firstValue) > 0 && thirdValue.CompareTo(secondValue) > 0 ||
            thirdValue.CompareTo(firstValue) >= 0 && thirdValue.CompareTo(secondValue) > 0 ||
            thirdValue.CompareTo(firstValue) > 0 && thirdValue.CompareTo(secondValue) >= 0)
        {
            return thirdValue;
        }
        return default;
    }

    11 references | 11/11 passing
    public T MaxMethod()
    {
        T max = GenericMaximum<T>.MaxValue(this.firstValue, this.secondValue, this.thirdValue);
        return max;
    }
}
```

UC 4

Extend the max method to also print the max to std out using Generic Method

- Write printMax Generic Method which is internally called from testMaximum

PrintArray Sample Code

```
25 references
public class GenericMaximum<T> where T : IComparable
{
    public T[] value;
    12 references | 11/11 passing
    public GenericMaximum(T[] value)
    {
        this.value = value;
    }
    1 reference
    public T[] Sort(T[] values)
    {
        Array.Sort(values);
        return values;
    }
    2 references
    public T MaxValue(params T[] values)
    {
        var sorted_values = Sort(values);
        return sorted_values[^1];
    }
    11 references | 11/11 passing
    public T MaxMethod()
    {
        var max = MaxValue(this.value);
        return max;
    }
    1 reference
    public void PrintMaxValue()
    {
        var max = MaxValue(this.value);
        Console.WriteLine("Maximum value is " + max);
    }
    0 references
    static void Main(string[] args)
    {
        int[] arr = { 112, 344, 432, 555, 678 };
        GenericMaximum<int> generic = new GenericMaximum<int>(arr);
        generic.PrintMaxValue();
    }
```

BridgeLabz
Employability Delivered

BridgeLabz

Employability Delivered

Thank You