**Generics**

**// It allows you to write a class or method that can work with any data type.**

**// We can create our own generic interfaces, classes, methods, events and delegates.**

**// A generic class or method can be defined using angle brackets**

class TestGenericClass<T>

{

//Generic with variable

private T variable1;

//Generic with constructor

public TestGenericClass(T value)

{

variable1 = value;

}

//Generic with member function

public T TestFunction(T param1)

{

Console.WriteLine("Parameter type: {0}, value: {1}", typeof(T).ToString(), param1);

Console.WriteLine("Return type: {0}, value: {1}", typeof(T).ToString(), variable1);

return variable1;

}

//Generic with property

public T property1 { get; set; }

}

**Implement:**

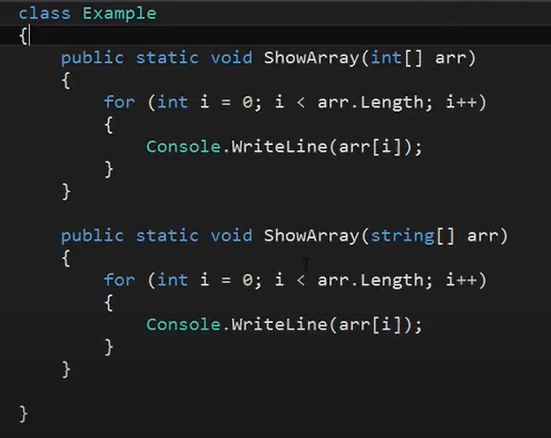
TestGenericClass<int> intGenericClass = new TestGenericClass<int>(10);

int val = intGenericClass.TestFunction(200);

TestGenericClass<string> stringGenericClass = new TestGenericClass<string>("Hello generic");

string val1 = stringGenericClass.TestFunction("Using generic");

**Example 1:**



Text

Description automatically generated

***The Problem:*** For printing array of each data types we had to create many overloading methods which unnecessarily increases the code.

**Solution**: To use generic, one versatile method for every arrays of different data types.

Define Type as anything let say as T so the syntax would be: *MethodName<T>(T[] array)*

Text

Description automatically generated

**Example 2:**

Graphical user interface, text, application

Description automatically generated with medium confidence

Graphical user interface, website

Description automatically generated

Solution:

Graphical user interface, text

Description automatically generated

Text

Description automatically generated