Laboratory 1

Theme: Generalization. Create a generalized method

***Generic***is a class which allows the user to define classes and methods with the placeholder. Generics were added to version 2.0 of the C# language. The basic idea behind using Generic is to allow type (Integer, String, … etc and user-defined types) to be a parameter to methods, classes, and interfaces. A primary limitation of collections is the absence of effective type checking. This means that you can put any object in a collection because all classes in the C# programming language extend from the object base class. This compromises type safety and contradicts the basic definition of C# as a type-safe language. In addition, using collections involves a significant performance overhead in the form of implicit and explicit type casting that is required to add or retrieve objects from a collection. To address the type safety issue, the [**.NET**](https://www.geeksforgeeks.org/c-net-framework-basic-architecture-component-stack/)framework provides generics to create classes, structures, interfaces, and methods that have placeholders for the types they use. Generics are commonly used to create type-safe collections for both reference and value types. The [**.NET**](https://www.geeksforgeeks.org/c-net-framework-basic-architecture-component-stack/) framework provides an extensive set of interfaces and classes in the System.Collections.Generic namespace for implementing generic collections.

**Generic Class**

Generics in C# is its most powerful feature. It allows you to define the type-safe data structures. This out-turn in a remarkable performance boost and high-grade code, because it helps to reuse data processing algorithms without replicating type-specific code. Generics are similar to templates in C++ but are different in implementation and capabilities. Generics introduces the concept of type parameters, because of which it is possible to create methods and classes that defers the framing of data type until the class or method is declared and is instantiated by client code. Generic types perform better than normal system types because they reduce the need for boxing, unboxing, and type casting the variables or objects. Parameter types are specified in generic class creation. **To create objects of generic class, following syntax is used:**

BaseType obj = new BaseType ()

**Example:**

* Csharp

|  |
| --- |
| // C# program to show working of  // user defined Generic classes  using System;    // We use < > to specify Parameter type  public class GFG<T> {        // private data members      private T data;        // using properties      public T value      {            // using accessors          get          {              return this.data;          }          set          {              this.data = value;          }      }  }    // Driver class  class Test {        // Main method      static void Main(string[] args)      {            // instance of string type          GFG<string> name = new GFG<string>();          name.value = "GeeksforGeeks";            // instance of float type          GFG<float> version = new GFG<float>();          version.value = 5.0F;            // display GeeksforGeeks          Console.WriteLine(name.value);            // display 5          Console.WriteLine(version.value);      }  } |

**Output :**

GeeksforGeeks

5

**Explanation:** The preceding example defines a generic class, *GFG*, which uses a generic type parameter ‘T’. In the Main() method, two instances of GFG have been created by replacing ‘T’ with ‘string’ and ‘float’ data types. These objects are used to store ‘string’ and ‘float’ values respectively. The GFG class ensures type safety by accepting the required type in its constructor. **A Generic method with various parameters:** Just as a method can take one argument, generics can take various parameters. One argument can be passed as a familiar type and other as a generic type, as shown below :

* **Example:**
* Csharp

|  |
| --- |
| // C# program to show multiple  // type parameters in Generics  using System;    public class GFG {        // Generics method      public void Display<TypeOfValue>(string msg, TypeOfValue value)      {          Console.WriteLine("{0}:{1}", msg, value);      }  }    // Driver class  public class Example {        // Main Method      public static int Main()      {            // creating object of class GFG          GFG p = new GFG();            // calling Generics method          p.Display<int>("Integer", 122);          p.Display<char>("Character", 'H');          p.Display<double>("Decimal", 255.67);          return 0;      }  } |

* **Output :**

Integer:122

Character:H

Decimal:255.67

**Features of Generics**

Generics is a technique that improves your programs in many ways such as:

* It helps you in code reuse, performance and type safety.
* You can create your own generic classes, methods, interfaces and delegates.
* You can create generic collection classes. The .NET framework class library contains many new generic collection classes in System.Collections.Generic namespace.
* You can get information on the types used in generic data type at run-time.

**Advantages of Generics**

* **Reusability:**You can use a single generic type definition for multiple purposes in the same code without any alterations. For example, you can create a generic method to add two numbers. This method can be used to add two integers as well as two floats without any modification in the code.
* **Type Safety:**Generic data types provide better type safety, especially in the case of collections. When using generics you need to define the type of objects to be passed to a collection. This helps the compiler to ensure that only those object types that are defined in the definition can be passed to the collection.
* **Performance:**Generic types provide better performance as compared to normal system types because they reduce the need for boxing, unboxing, and typecasting of variables or objects.

Generics in C# is a feature that allows for the creation of reusable code by creating parameterized types. In simple terms, it enables us to create classes, interfaces, and methods that work with different data types without having to define the data type explicitly.

1. By using generics, we can write code that is more flexible and less prone to errors. For example, instead of writing a separate method for each data type, we can write a single method that can work with different types of data.
2. The main benefit of using generics is that it provides type safety. When we define a generic class, interface or method, we can specify the type parameter that it can work with. This ensures that the code will only accept and operate on data types that match the specified type parameter, preventing errors that can occur due to data type mismatches.
3. Generics also help in reducing code duplication, as it is possible to write a generic class or method that can work with any data type, instead of writing multiple methods or classes for different data types.
4. C# provides a number of built-in generic types, such as List<T>, Dictionary<TKey, TValue>, and Nullable<T>. We can also create our own generic classes, interfaces, and methods by using the syntax <T>, where T is the type parameter that can be replaced with any valid data type.