Answer on Question #44686, Programming, C#

Task: Which features of C# will allow to reuse the existing application code? Describe in brief. Define an interface. Briefly describe the benefits of using interfaces.

Solution:

Generics introduce to the .NET Framework the concept of type parameters, which make it possible to design classes and methods that defer the specification of one or more types until the class or method is declared and instantiated by client code. For example, by using a generic type parameter T you can write a single class that other client code can use without incurring the cost or risk of runtime casts or boxing operations, as shown here:

```
// Declare the generic class.
public class GenericList<T>
{
   void Add(T input) { }
}
class TestGenericList
{
   private class ExampleClass { }
   static void Main()
   {
      // Declare a list of type int.
      GenericList<int> list1 = new GenericList<int>();

      // Declare a list of type string.
      GenericList<string> list2 = new GenericList<string>();

      // Declare a list of type ExampleClass.
      GenericList<ExampleClass> list3 = new GenericList<ExampleClass>();
   }
}
```

Generics Overview

- Use generic types to maximize code reuse, type safety, and performance.
- The most common use of generics is to create collection classes.
- The .NET Framework class library contains several new generic collection classes in the System.Collections.Generic namespace. These should be used whenever possible instead of classes such as ArrayList in the System.Collections namespace.
- You can create your own generic interfaces, classes, methods, events and delegates.
- Generic classes may be constrained to enable access to methods on particular data types.
- Information on the types that are used in a generic data type may be obtained at run-time by using reflection.

An interface contains only the signatures of methods, properties, events or indexers. A class or struct that implements the interface must implement the members of the interface that are specified in the interface definition. In the following example, class ImplementationClass must implement a method named SampleMethod that has no parameters and returns void.

```
interface ISampleInterface
{
   void SampleMethod();
}

class ImplementationClass : ISampleInterface
{
   // Explicit interface member implementation:
   void ISampleInterface.SampleMethod()
   {
      // Method implementation.
   }

   static void Main()
   {
      // Declare an interface instance.
      ISampleInterface obj = new ImplementationClass();

      // Call the member.
      obj.SampleMethod();
   }
}
```

An interface can be a member of a namespace or a class and can contain signatures of the following members:

- Methods
- Properties
- Indexers
- Events

An interface can inherit from one or more base interfaces.

When a base type list contains a base class and interfaces, the base class must come first in the list. A class that implements an interface can explicitly implement members of that interface. An explicitly implemented member cannot be accessed through a class instance, but only through an instance of the interface.

Why Use Interfaces?

- To allow a class to inherit multiple behaviors from multiple interfaces.
- To avoid name ambiguity between the methods of the different classes
- To combine two or more interfaces such that a class need only implement the combined result.

- To allow Name hiding. Name hiding is the ability to hide an inherited member name from any code outside the derived class.