

Generics

Generic types

- Design classes without specifying the types at definition stage
- Design methods without specifying the types for parameters at definition stage
- Use a placeholder at definition stage that will be replaced by type during instantiation
- Enable type-safe coding
- Increases performance due to reduction in conversions, boxing/unboxing

Generic class

Note that T is available to the nested **Node** class. When GenericList<T> is instantiated with a concrete type, for example as a GenericList<int>, each occurrence of T will be replaced with int.

```
// type parameter T in angle brackets
public class GenericList<T>
{
    // The nested class is also generic on T.
    private class Node
    {
        // T used in non-generic constructor.
        public Node(T t)
        {
            next = null;
            data = t;
        }

        private Node next;
        public Node Next
        {
            get { return next; }
            set { next = value; }
        }
    }

    // T as private member data type.
    private T data;

    // T as return type of property.
    public T Data
    {
        get { return data; }
        set { data = value; }
    }
}

private Node head;

// constructor
public GenericList()
{
    head = null;
}
```

```
// T as method parameter type:
public void AddHead(T t)
{
    Node n = new Node(t);
    n.Next = head;
    head = n;
}
```

The following code example shows how client code uses the generic `GenericList<T>` class to create a list of integers. Simply by changing the type argument, the following code could easily be modified to create lists of strings or any other custom type:

```
class TestGenericList
{
    static void Main()
    {
        // int is the type argument
        GenericList<int> list = new GenericList<int>();

        for (int x = 0; x < 10; x++)
        {
            list.AddHead(x);
        }

        foreach (int i in list)
        {
            System.Console.Write(i + " ");
        }
        System.Console.WriteLine("\nDone");
    }
}
```

Generic methods

A generic method is a method that is declared with type parameters, as follows:

```
static void Swap<T>(ref T lhs, ref T rhs)
{
    T temp;
    temp = lhs;
    lhs = rhs;
    rhs = temp;
}
```

The following code example shows one way to call the method by using `int` for the type argument:

```
public static void TestSwap()
{
    int a = 1;
    int b = 2;

    Swap<int>(ref a, ref b);
    System.Console.WriteLine(a + " " + b);
}
```

```
}
```

You can also omit the type argument and the compiler will infer it. The following call to `Swap` is equivalent to the previous call:

```
Swap(ref a, ref b);
```