String

The string is immutable, Immutable means if you create a string object then you cannot modify it and It always creates a new object of string type in memory.

Example

```
string strMyValue = "Hello Visitor";

// create a new string instance instead of changing the old one

strMyValue += "How Are";

strMyValue += "You ??";
```

C#

StringBuilder

StringBuilder is mutable, which means if create a string builder object then you can perform any operation like insert, replace, or append without creating a new instance every time. it will update the string in one place in memory and doesn't create new space in memory.

Example

```
StringBuilder sbMyValue = new StringBuilder("");

sbMyValue.Append("Hello Visitor");

sbMyValue.Append("How Are You ??");

string strMyValue = sbMyValue.ToString();
```

Boxing and Unboxing in C#

Boxing and unboxing in C# allow developers to convert .NET data types from value type to reference type and vice versa. Converting a value type to a reference type is called boxing in C# and converting a reference type to a value type is called unboxing in C#.

C# provides a "unified type system". All types including value types derive from the type object. It is possible to call the object methods on any value, even values of "primitive" types, such as int. The

example is shown below.

```
using System;

class Test
{
   static void Main()
   {
      Console.WriteLine(3.ToString());
   }
}
```

It calls the object-defined ToString method on an integer literal. The example.

```
class Test
{
   static void Main()
   {
      int i = 1;
      object o = i; // boxing
      int j = (int)o; // unboxing
   }
}
```

An int value can be converted into an object and back again into an int.

This example shows both, boxing and unboxing. When a variable of a value type needs to be converted into a reference type, an object box is allocated to hold the value, and the value is copied into the box.

Unboxing is just the opposite. When an object box is cast back to its original value type, the value is copied out of the box and into the appropriate storage location.

Boxing conversions

A boxing conversion permits any value type to be implicitly converted to the type object or to any interface type implemented by the value type. Boxing a value of a value-type consists of allocating an object instance and copying the value-type value into that instance.

For example, for any value-type G, the boxing class would be declared as follows.

```
class VBox
{
   G value;
   G_Box(G g)
   {
      value = g;
   }
}
```

For example

```
int i = 12;
object box = i;
if (box is int)
{
```

```
    Console.Write("Box contains an int");
}
```

C#

The above code will output the string "Box contains an int" on the console.

A boxing conversion implies making a copy of the value being boxed. This is different from a conversion of a reference type to a type object, in which the value continues to reference the same instance and simply is regarded as the less derived type object.

For example, given the declaration.

```
struct Point
{
    public int x, y;
    public Point(int x, int y)
    {
        this.x = x;
        this.y = y;
    }
}
```

C#

the following statements.

```
Point p = new Point(10, 10);
object box = p;
p.x = 20;
```

```
Console.Write(((Point)box).x);
```

C#

These will output the value 10 on the console because the implicit boxing operation that occurs in the assignment of p to box causes the value of p to be copied. Had Point instead been declared a class, the value 20 would be output because p and box would reference the same instance.

Unboxing conversions

An unboxing conversion permits an explicit conversion from a type object to any value type or from any interface type to any value type that implements the interface type. An unboxing operation consists of first checking that the object instance is a boxed value of the given value, and then copying the value out of the instance. Unboxing conversion of an object box to a value-type G consists of executing the expression ((G_Box)box).value.

Thus, the statements,

```
object box = 12;

int i = (int)box;
```

C#

conceptually correspond to,

```
object box = new int_Box(12);

int i = ((int_Box)box).value;
```

C#

For an unboxing conversion to a given value type to succeed at run-time, the value of the source argument must be a reference to an object that was previously created by boxing a value of that value type. If the source argument is null or a reference to an incompatible object, an InvalidCastException is thrown.

Conclusion

This type-system unification provides value types with the benefits of object-ness without introducing unnecessary overhead. For programs that don't need int values to act like objects, int values are simply 32-bit values. For programs that need int values to behave like objects, this capability is available on demand. This ability to treat value types as objects bridges the gap between value types and reference types that exist in most languages.

https://www.c-sharpcorner.com/blogs/difference-between-string-and-stringbuilder-in-c-sharp1

https://www.c-sharpcorner.com/article/boxing-and-unboxing-in-C-Sharp/

https://dotnettutorials.net/lesson/boxing-and-unboxing-in-csharp/

https://www.javatpoint.com/type-of-assembly-in-c-sharp

https://medium.com/@devmaleeq/c-assemblies-compiler-and-everything-in-between-8697c3e6c496

https://www.w3resource.com/csharp-exercises/

Hands-on Practice:

```
using System;
using System.Text;
Console.Write("Enter your account number = ");
int acno = Convert.ToInt32(Console.ReadLine());

Console.Write("Enter your name = ");
string name = Console.ReadLine();

Console.WriteLine("Your account number:" + acno);
Console.WriteLine("Your Name:" + name);

//string
string strMyValue = "Hello Visitor";
```

```
Console.WriteLine(strMyValue);
//create a new string instance instead of changing the old one
strMyValue += "How Are";
Console.WriteLine(strMyValue);
strMyValue += "You ??";
Console.WriteLine(strMyValue);

//stringbuilder
StringBuilder sbMyValue = new StringBuilder("");
sbMyValue.Append("Hello Visitor");
Console.WriteLine(sbMyValue);
sbMyValue.Append("How Are You ??");
Console.WriteLine(sbMyValue);
string strMyValue1 = sbMyValue.ToString();
Console.WriteLine(strMyValue);
```