Boxing, and Unboxing

.NET CORE

Subtyping between more complex ***types*** *relates to subtyping between their components. If type* **Cat** *is a subtype of* **Mammal**, *then an expression of type* **Cat** *should be substitutable wherever an expression of type* **Mammal** *is used.*

HTTPS://DOCS.MICROSOFT.COM/EN-US/DOTNET/CSHARP/PROGRAMMING-GUIDE/TYPES/BOXING-AND-UNBOXING

Boxing

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/types/boxing-and-unboxing#boxing

Unboxing

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/types/boxing-and-unboxing#unboxing

The concept of ***boxing*** and ***unboxing*** underlies the C# ***Unified Type System*** in which a value of any ***type*** can be treated as an object.

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/types/boxing-and-unboxing

Boxing happens when any ***value*** type is cast to an ***object***. The value is wrapped to give it reference type behavior.

```
int i = 123;
// The following line boxes i.
object o = i;
```

BOXING is implicit

Unboxing extracts the ***value*** type from the object.

```
o = 123;
i = (int)o;  // unboxing
```

UNBOXING is explicit

Boxing to the object type allows different types to inhabit the same array.

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/types/boxing-and-unboxing

```
// String.Concat example.
// String.Concat has many versions. Rest the mouse pointer on
// Concat in the following statement to verify that the version
// that is used here takes three object arguments. Both 42 and
// true must be boxed.
Console.WriteLine(String.Concat("Answer", 42, true));

// List example.
// Create a list of objects to hold a heterogeneous collection
// of elements.
List<object> mixedList = new List<object>();

// Add a string element to the list.
mixedList.Add("First Group:");

// Add some integers to the list.
for (int j = 1; j < 5; j++)
{
    // Rest the mouse pointer over j to verify that you are adding
    // an int to a list of objects. Each element j is boxed when
    // you add j to mixedList.
    mixedList.Add(j);
}
```

```
// Add another string and more integers.
mixedList.Add("Second Group:");
for (int j = 5; j < 10; j++)
{
    mixedList.Add(j);
}

// Display the elements in the list. Declare the loop variable by
// using var, so that the compiler assigns its type.
foreach (var item in mixedList)
{
    // Rest the mouse pointer over item to verify that the elements
    // of mixedList are objects.
    Console.WriteLine(item);
}

// The following loop sums the squares of the first group of boxed
// integers in mixedList. The list elements are objects, and cannot
// be multiplied or added to the sum until they are unboxed. The
// unboxing must be done explicitly.
var sum = 0;
for (var j = 1; j < 5; j++)
{
    // The following statement causes a compiler error: Operator
    // '*' cannot be applied to operands of type 'object' and
    // 'object'.
    //sum += mixedList[j] * mixedList[j]);

    // After the list elements are unboxed, the computation does
    // not cause a compiler error.
    sum += (int)mixedList[j] * (int)mixedList[j];
}
```

Output

```
// The sum displayed is 30, the sum of 1 + 4 + 9 + 16.
Console.WriteLine("Sum: " + sum);

// Output:
// Answer42True
// First Group:
// 1
// 2
// 3
// 4
// Second Group:
// 5
// 6
// 7
// 8
// 9
// Sum: 30
```

Unboxing

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/types/boxing-and-unboxing#unboxing

Unboxing is an explicit conversion from the type ***object*** to a ***value*** type or from an ***interface*** type to a ***value*** type that implements the interface. An ***unboxing*** operation consists of:

- Checking the object instance to make sure that it is a boxed value of the given value type.
- Copying the value from the instance into the value-type variable.

```
int i = 123;        // a value type
object o = i;       // boxing
int j = (int)o;     // unboxing
```

```
class TestUnboxing
{
    static void Main()
    {
        int i = 123;
        object o = i;  // implicit boxing

        try
        {
            int j = (short)o;  // attempt to unbox

            System.Console.WriteLine("Unboxing OK.");
        }
        catch (System.InvalidCastException e)
        {
            System.Console.WriteLine("{0} Error: Incorrect unboxing.", e.Message);
        }
    }
}
```

```
Specified cast is not valid. Error: Incorrect unboxing.
```