Boxing & Unboxing in C#

Boxing

Boxing is the process of converting a value type (such as an int, double, or struct) into an object. When a value type is boxed, it is wrapped inside a System.Object and stored on the heap instead of the stack. This allows the value type to be used where an object is required, such as in collections that store objects (like ArrayList).

Example of Boxing:

```
int number = 123;        // Value type
object obj = number;     // Boxing
```

Unboxing

Unboxing is the reverse process of boxing. It involves converting an object back into a value type. Unboxing extracts the value type from the object. It's important to ensure that the object being unboxed actually contains the correct value type, otherwise, an InvalidCastException will be thrown.

Example of Unboxing:

```
object obj = 123;        // Boxing
int number = (int)obj;   // Unboxing
```

Important Points

1. Performance Impact: Boxing and unboxing are computationally expensive operations because they involve copying data between the stack and the heap. Excessive boxing and unboxing can lead to performance issues.
2. Type Safety: While unboxing, you must explicitly cast the object to the appropriate value type. If the cast is invalid, it will throw an exception.

Practical Example:

Consider a scenario where you need to store various value types in an ArrayList (which can store objects):

```
ArrayList list = new ArrayList();

int number = 42;
list.Add(number);     // Boxing

// Later retrieval
int retrievedNumber = (int)list[0];   // Unboxing
```

In this example:
- The integer number is boxed when added to the ArrayList.
- The object stored in the ArrayList is unboxed back to an integer when retrieved.