GENERIC PROGRAMMING

Boxing and Unboxing

- Boxing is a method in which a value type is converted into object(reference type).
- Eg:

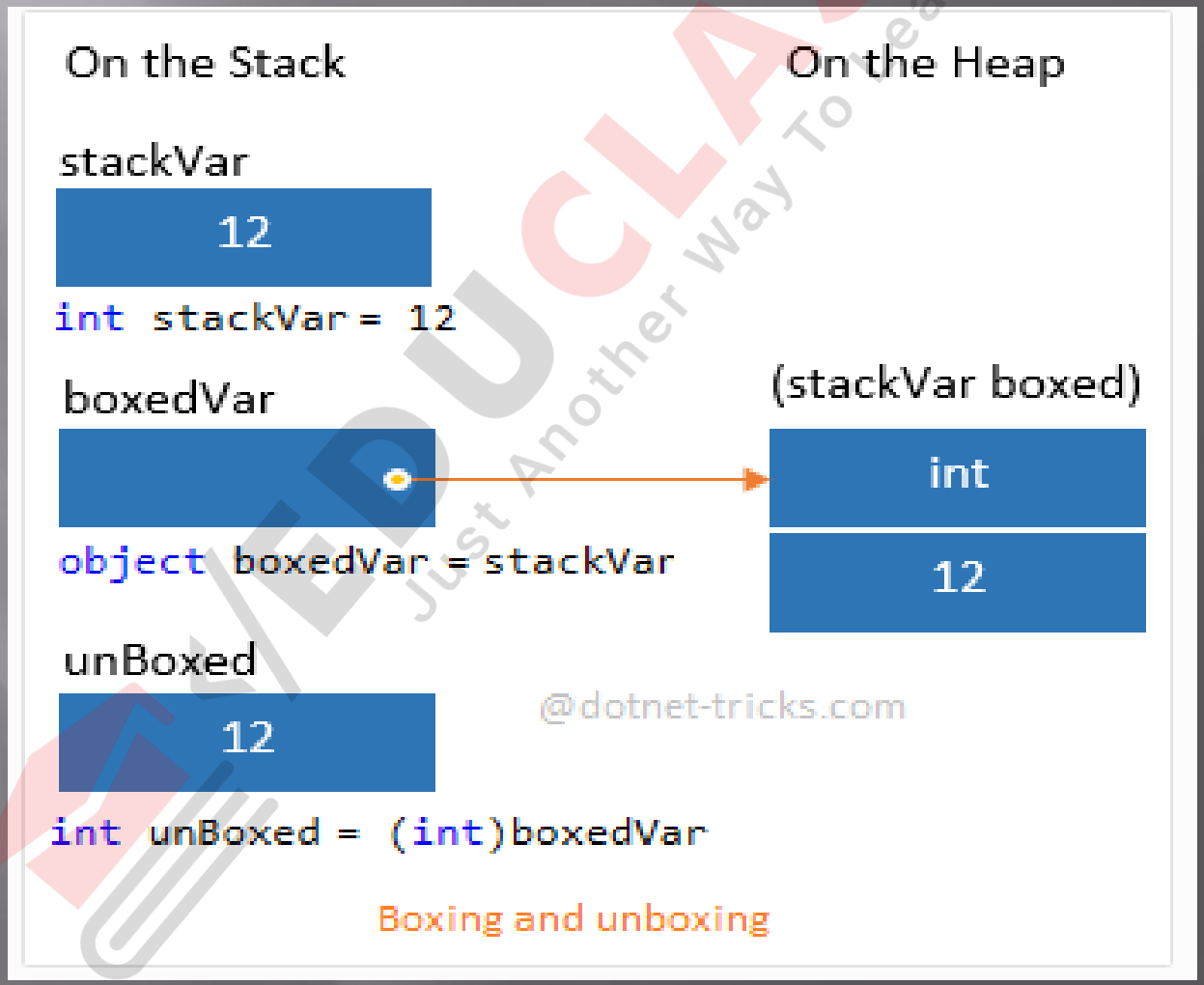
```
int i = 30;
object o=i;
```

- When the CLR boxes a value type, it wraps the value inside a System.Object and stores it on the managed heap.
- Boxing is implicit.

- UnBoxing, on the other hand, is a term used for conversion of object to a value type.
- Eg:

```
int i = (int) o;
```

- UnBoxing is explicit.


On the Stack
On the Heap
stackVar
12
int stackVar = 12
boxedVar
(stackVar boxed)
int
12
object boxedVar = stackVar
unBoxed
12
@dotnet-tricks.com
int unBoxed = (int)boxedVar
Boxing and unboxing

Drawbacks

- Not type safe.
- Performance degradation due to conversion from value type to reference type..

Generics

- Generics is introduced in C# 2.0.
- The most common use of generics is to create collection classes.
- Use generic types to maximize code reuse, type safety, and performance.
- Generic classes may be constrained to enable access to methods on particular data types.

Generic types

- A generic type declares *type parameters – placeholder types to be filled in by the*
- consumer of the generic type supplies the *type arguments.*

Generic Methods

- A generic method declares type parameters within the signature of a method.
- Generally, there is no need to supply type arguments to a generic method, because the compiler can implicitly infer the type.
- Type parameters can be introduced in the declaration of classes, structs, interfaces, delegates, and methods.
- Other constructs, such as properties, indexers, events, fields, constructors, operators cannot *introduce a type parameter, but can use one.*

Generic Class

```
public class Stack<T>
{
   int position;
   T[] data = new T[100];
   public void Push (T obj)   { data[position++] = obj;  }
   public T Pop()             { return data[--position]; }
}
```

```
Stack<int> stack = new Stack<int>();
stack.Push(5);
stack.Push(10);
int x = stack.Pop();          // x is 10
int y = stack.Pop();          // y is 5
```

- Generics provide type safety.
- Generics eliminates boxing and unboxing.
- There is no need to write code to test for the correct data type because it is enforced at compile time. The need for type casting and the possibility of run-time errors are reduced.
- By providing strong typing, a class built from a generic lets visual studio provide IntelliSense.
- Generic collection types generally perform better for storing and manipulating value types because there is no need to box the value types.