

Re: Value Types and Reference Types

```
int x;
```

gives a value type x.

```
System.Int32 x;
```

will give a reference type x;

Basically the compiler says "Oh, x is a value type so we can optimize it and put it on the stack". If you do `int x` and then `object y = (object)x`. Then the compiler will convert x into an object (after all, its base type is an object since they all derive from it). This is slow cause now the compiler has to create the object on the heap and its called boxing.

I think if you think of it as an optimization of special types then its not a difficult concept. Basically using the special keywords you tell the compiler they are value types... which just means that they are to be optimized and put on the stack rather than the heap. And remember that if you convert from optimized to non-optimized or vice versa that there is a penalty.

Other than that they pretty much are exactly the same. Again, `int x` and `Int32 x`; are pretty much identical except for speed/memory issues. The heap and stack are just memory. An object has more overhead of course and thats why value types are faster. No reason to have overhead for something like

```
int x = 3;  
int y = 9;  
int z = x * y;
```

The compiler will simply do that in the most optimized way which is using the ALU of the cpu (done in 1-2 instructions in assembly).

if you do

```
Int32 x = 3;  
Int32 y = 9;  
Int32 z = x * y;
```

then the compiler will treat them as full blown objects and have to find the operator *, call the constructors, allocate memory, etc... Theres no need for that. (although its a simple example so...)

Hope that helps,
Jon

