**CLR via C# - Jeff Richter**

**Part 2 Designing Types**

**Type Fundamentals**

**Object.ToString()** is expected to be aware of the **CultureInfo** associated with the calling thread.

Object.**GetType()** is non virtual so type safety cant be violated.

When new is called on an object.

1. # of bytes is calculated for all instance fields in type and in its base types up to Object. Also needed is bytes for type pointer and sync block index.

2. Allocates memory from managed heap and sets bytes to zero.

3. Inits objects type pointer and sync block index members.

4. Calls objects constructor

4. Returns a reference(pointer) to object.

**Casting between types**

At runtime CLR always knows what type an object is.

A type can **implicitly** cast to any of its **base types**. However it must **explicitly** cast to its **derived types**.

CLR types checks at runtime. This impacts performance due to having to walk the inheritance chain to find the proper type.

As a result use **as** to try and cast without causing an exception if cast fails

Employee e = o as Employee;

if (e != null) { …}

**Assemblies**

When checking for a type’s definition, the compiler must be told which assemblies to examine by using the **/reference** compiler switch. After the compiler finds the proper assembly, the assembly information and the type information is emitted into the resulting managed module’s metadata.

void M3() {

Employee e;

Int32 year;

e = new Manager();

e = Employee.Lookup("Joe");

year = e.GetYearsEmployed();

e.GetProgressReport();

}

As the just-in-time (JIT) compiler converts M3’s Intermediate Language (IL) code into native CPU instructions, it notices all of the types that are referred to inside M3: Employee, Int32, Manager, and String (because of "Joe"). At this time, the CLR ensures that the assemblies that define these types are loaded. Then, using the assembly’s metadata, the CLR extracts information about these types and creates some data structures to represent the types themselves.

