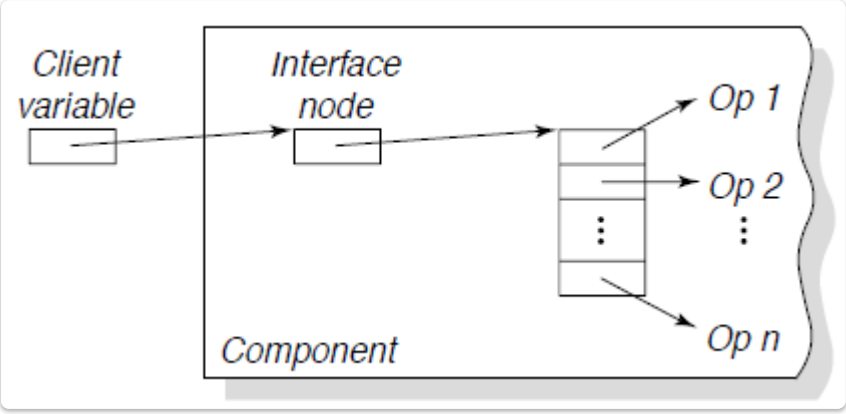
07 - .NET Framework

COM

Before .NET, there was COM (Component Object Model), which was a binary standard for interfaces, specifying an object model and programming requirements that enable COM components to interact. It is language independent: the only requirements are that the language can create structures of pointers and - either explicitly or implicitly - call functions through pointers.



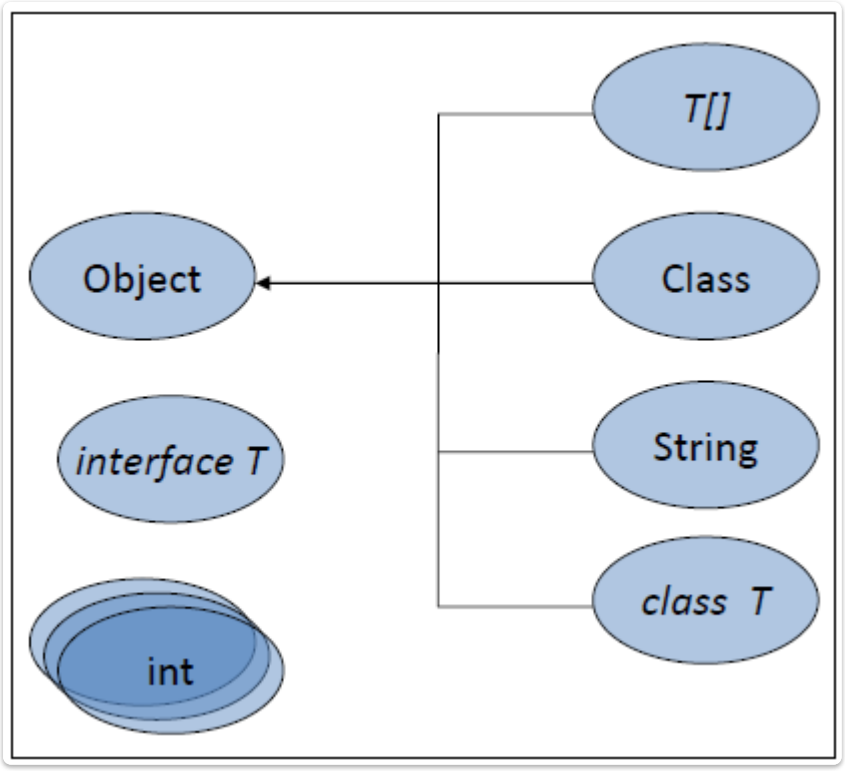
.NET

The .NET Framework is the Microsoft approach to components. They wanted to achieve integration and interoperability of applications that are distributed (logically and physically) on heterogeneous nodes (i.e. applications written in different languages). It provides support for rapid development of XML web services and applications. It consists of:

- Common Language Specification (CLS): guidelines that language should follow to communicate with other .NET languages. The rules specified by the CLS makes possible for .NET to uniform the type systems of the supported languages;
- Base Class Library (BCL): a consistent, object-oriented library, containing reusable types and utilities to develop CLI, GUI and more (kind of Java API). It is organized by namespaces (in Java API -> packages), which consist of many classes and sub-namespaces. Developers can create custom namespaces;
- Common Language Runtime (CLR): virtual machine environment for development and execution (kind of JVM). Thanks to CLR, a class in one language can inherit properties and methods from related classes in other languages. Programs are first JIT compiled into a Common Intermediate Language (CIL), which is then compiled into target native code based on the architecture (note that this is different from the bytecode in Java, as CIL is not interpreted, but just re-compiled again). The IL code can be in the format of .exe or DLL, and when generated by the .NET compiler they are called managed code.

Type system

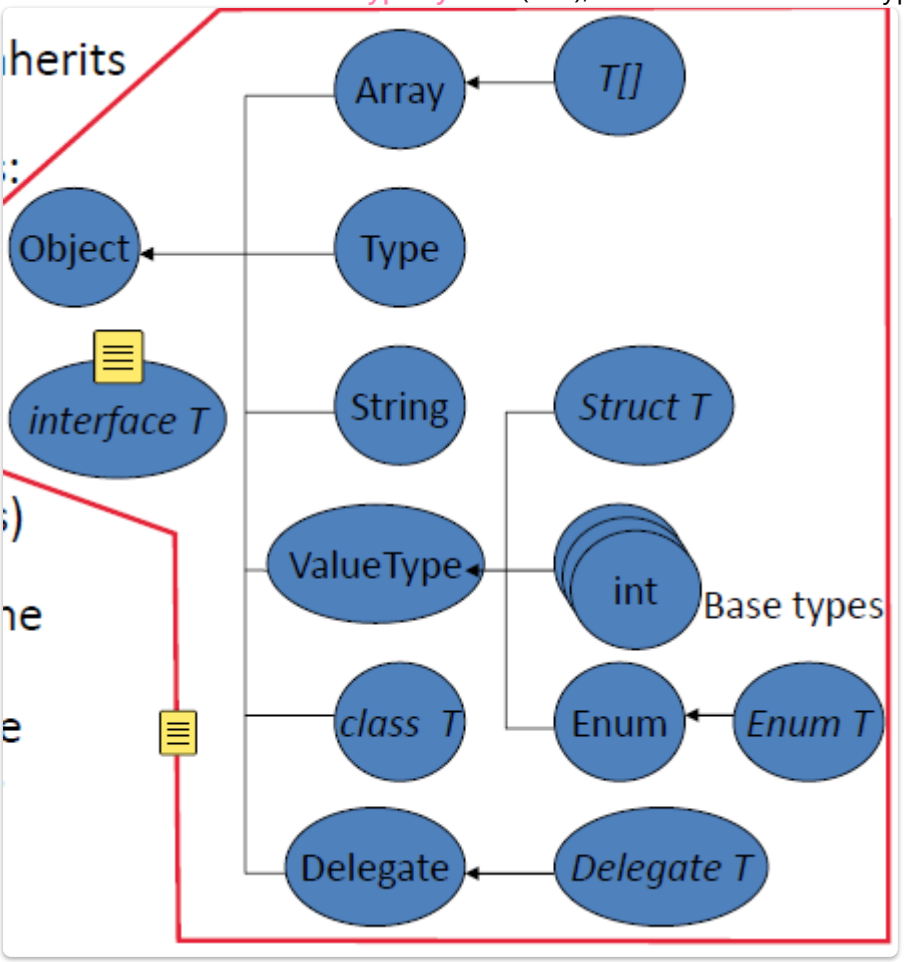
In Java, the type system is defined as follows:



Object is the indirect superclass of every class, but are not related to interfaces and primitive types. Then, there are 2 type constructors:

- Array;
- Class, which can either be class objects (user defined) or class types (generated by the JVM, e.g. arrays of type T).

The CLR defines a Common Type System (CTS), a standard set of data type and rules for creating new types.



Differently from Java's type system, we have 3 type constructors:

- Enum: used for constants;

- Struct: like class, but no inheritance (they only inherit from Object) and they are allocated on the stack (not heap like Objects);
- Delegate: represents references to methods with a specific parameter list and return type. They can be used to pass methods as arguments to other methods, thus supporting a functional programming style (higher-order features). They can also be used to support event-based programming, where event handlers are invoked through delegates: CLR provides multicast delegates - kind of delegate that holds a list of delegate - to support notification to many listeners. Delegates are a pair (env, func) like closures, but they are not equivalent, because in this case the env must be an object of the same class in which the function is declared. Note that in Java there are no delegates, so interfaces (Listener for event-based programming) are used instead.

Then, we have Value Types which contains base types (numbers) and structs, but once again they are not stores on the heap. However, when a value type is upcasted to Object, it is boxed in a wrapper on the heap. The reverse operation is called unboxing.

Component Model (Intermediate Language) of .NET

Assemblies are the components of .NET. An assembly is a single, pre-compiled and self-described CIL module (CIL code and its metadata). It is built from one or more classes/modules and deployed in a DLL assembly file.

Characteristics:

- Self-describing: it needs to be executable;
- Platform-independent;
- Can be located by querying its strong name: (publisher token, assembly name, version vector, culture), where the version vector = (major, minor, build, patch).

An assembly consists of up to 4 parts:

- Manifest: self-description of the component, it contains strong name, type reference information, list of files present, information on references assemblies references and more;
- Metadata: modules' metadata;
- CIL code: modules' CIL code;
- Resources: not executable information;

Assemblies can either be a .DLL or .EXE. The difference is that a .DLL is not executable (just like a class file), while the .EXE can be executed:

- PE (Portable Executable) is the standard MS format for executable files;
- PE .NET causes a call to the CLR runtime at the beginning.

In addition, a .DLL component can be deployed:

- As a private component (knowing the target client);
- As a shared public component, by publishing it in a centralized repository named Global Assembly Cache (GAC), typically using its strong name.

Connection Model of .NET

Composition can either happen by:

- Aggregation: one object has a reference to another object, but the second object can exist independently of the first object. So, it’s less effort for the outer object. On the other hand, the outer object can’t specialize the behaviour of the inner object;

- Containment: one object owns or contains another object, and the second object cannot exist independently of the first object. Only the outer object is visible externally, but the outer object can use inner and enforce controls, acting as a proxy.

Remoting Connectors for .NET Distributed Components

As 2 Java programs can't communicate if they are on different JVM, it is the same if two programs run on different CLR.

To allow them to communicate, there is the Remoting Channel Connection. This is done through marshalling:

- Marshal By Value (MBV): pass a copy of the object;
- Marshal By Reference (MBR): creates a proxy of a remote object. This is needed when a remote component must run at a remote site.

For asynchronous call-backs, there are Remoting Delegates, which can be invoked remotely for notification purposes.