1. Assemblies are auto-descriptive Packages of Types (class, enums, etc.)
2. Namespaces (and ‘using’ keyword use to avoid using the full name of Types in code).
3. ValueType vs. ReferenceType
   1. Boxing (creating an object around an Int32) and Unboxing (extracting the ValueType from the Box) is done automatically.
4. Fields/Methods : static vs. Instance.
5. Unit tests (Arrange/Act/Assert)
6. Object paradigm :
   1. Inheritance
      1. Layout of instances in memory.
      2. Instance Methods actually are static methods that accepts an implicit ‘this’ parameter.
   2. GetType() that gives the running instance type.
      1. using ‘is’ and ‘as’ keyword to test the running type (this respects the Liskov Substitution Principle)
   3. virtual/override
   4. Virtual Method Tables (a call to a virtual method occurs one indirection based on the running type of the instance)
   5. ‘sealed’ keyword (to forbid override)
7. Interface are Contracts
   1. Abstract classes can offer base implementation (but recall that a class can only have one base class).
   2. Interface members can be explicitely implemented.
      1. Enables support of different returned types (for identical parameters)
      2. Enables to « close » an implementation (like ‘sealed’ keyword can do).
8. IDisposable
   1. Acquire & Release as soon as possible
9. ‘using’ keyword is a syntactic sugar that guaranties try {…} finally { Dispose ! }
10. IEnumerable & IEnumerator
    1. ‘foreach’ keyword is a syntactic sugar for GetEnumerator/MoveNext/Current/Dispose.
11. Generics
12. IEnumerable<T> implementation
13. Dictionary<T> implementation
    1. Object.GetHashCode()
    2. Object.Equals
14. Design Pattern « Strategy » : IEqualityComparer<T>
15. Linked List (« Single Linked List » for our ITIDictionary<TKey,TValue>)
16. Yield return
17. MiniLinq : Extension methods at work
18. Func<T> , Action<T>, etc.
19. Linq in action : Ordering & Grouping
20. Observer pattern
    1. With interfaces and manual registration/unregistration
    2. With delegates
       1. Delegate.Combine does the dirty work : a combined delegate IS\_A delegate…
    3. With standard event support
       1. Standard signature : object source, EventArgs e
       2. ‘event’ keyword protects the delegate (only +=/-= are exposed)