# C# Language Reference

| | |
|---|---|
| Hello World | `class HelloWorld`<br>`{`<br>`  static void Main()`<br>`  {`<br>`  System.Console.WriteLine("Hello World");`<br>`  }`<br>`}` |
| Comments | `// - in line`<br>`/*  */ - section comment` |
| Namespaces | `Equivalent to Java packages`<br>`using *namespace*; at start of code to include namespace (not class).`<br>`Must include using System for most classes.` |
| Types | `Value types (eg int) dtored on stack`<br>`Reference types (eg String) stored on heap` |
| Built in value types | `byte, char, bool, sbyte, short, ushort, int, uint, float, double, decimal, long, ulong.`<br>`Can cast automatically or explicitly eg x = (short)y` |
| Variables | `eg int x = 1; Variables must be initialised before use.` |
| Constants | `eg const int x =1;` |
| Enumerations | `Base type defaults to int`<br>`eg`<br>`public enum Sizes`<br>`  {Small=1, Reg=2, Large=3}`<br>`Reference as Sizes.Small.`<br>`Can leave values out -> default will be 0, 1, 2 …` |
| Strings | `eg string s = "ABC"` |
| Case | `C# is case sensitive`<br>`use camelNotation for variables (eg int someName)`<br>`use PascalNotation for classes/methods (eg SomeMethod)` |
| if .. else | `if (expr) { } else { }` |
| switch | `switch (expr)`<br>`{ case expr: statement; break or goto;`<br>`  default: statement`<br>`}`<br>`will only fall thru a case statement if it is blank.  Does not default to fallthru without break.`<br>`Can switch on string expressions` |
| loops | `  while (expr is true)`<br>`  { }`<br>`or`<br>`  do`<br>`  { } while (expr is true)`<br>`or`<br>`  for (int i=start;i<end;i++)`<br>`  { }`<br>`or`<br>`  foreach (obj x in coll)`<br>`  {  x.blah(); }` |
| break/continue | `continue causes execution to return to top and continue`<br>`break ceases excution of the loop` |

| | |
|---|---|
| Operators | Assignment (=)<br>Arithmetic (+, -, *, /, %(modulus))<br>Increment (++, +=, +* etc), decrement (--)<br>Relational (==, !=, >, >=, < , <=)<br>Conditional(&&, ||, ! -> Note C# will short circuit expressions)<br>Logical (&, ^, |)<br>Ternary (cond-expr ? expr1 : expr2) |
| Preprocessor | #define, #if etc<br>#region name ->#endregion – marks a block of collapsible code |
| Classes | public class Ade:*base-class*<br>{ }<br>Ade a = new Ade(); |
| Access Modifiers | public : no restrictions<br>private : only accessible to class<br>protected : only accessible to class and subclasses<br>internal : accessible to any class in assembly<br>protected internal : == protected or internal |
| Methods | return-type Name(params)<br>{ } |
| Constructors | Same name as class and no return type. Can have multiple constructors with different param lists.<br>Copy constructor must be created manually by passing an object in to a constructor method.<br>A static constructor will run before any instance of the class is created |
| Destructor | Should only be used if there are unmanaged resources<br>Called by garbage collector<br>~ClassName()<br>{ } |
| Dispose | Can define a Dispose method – implement interface IDisposable.<br>Should suppress GC using GC.SuppressFinalize(this);<br>Called automaticalling in using clauses eg<br>using (x = new XYZ())<br>{ } Dispose called automatically. |
| Within class reference | this is the current object<br>base is the super class object |
| Static members | Belong to and referenced by the class name<br>Cannot be referenced using an object instance |
| Params – by reference/by value | Default is by value for value types<br>Use (ref int x) to pass by reference<br>values must be assigned a vlue before use. If not initially assigned then use out: (out int x) |
| Overloading methods | Must change types or number of parameters – just chaging return type doesn't work. |
| Properties | Make instance variables private – access is via properties<br>public int Xyz<br>{ get { return Xyz; }<br>set { Xyz = value } }<br>get or set are optional<br>Can then use property as if it were a normal variable.<br>eg a.Xyz++ |
| Inheritance | To override a base class method<br>base class must define method as virtual<br>public virtual void open()<br>to override it in child class<br>public override void open()<br>All methods are final by default.<br>Helps in versioning, eg add a new method in base class that has already been declared in a subclass. |

| | |
|---|---|
| | If method in subclass is the same as a virtual base method must use new to indicate it is not an override<br>eg public new virtual Xyz()<br>Use sealed keyword to make a class final so it can't be inherited |
| Abstract class/method | abstract public void Add();<br>Must be overriden by sub class.<br>Base class must also be abstract<br>abstract public class AdeBase { } |
| System.Object | Provides Equals(), GetHashCode(), GetType(), ToString(), Finalize(), MemberwiseClone(), ReferenceEquals() |
| Boxing/Unboxing | Boxing converts a value type to a reference type and is automatic<br>eg int I =123; i.ToString();<br>Unboxing converts from object to a value type – must be explicit<br>eg int I = 123; Object o = i; int j = (int)o; |
| Nesting classes | Can create private classes within a class. Use internal keyword. Similar to java static inner classes. If class is defined as public then it must be referenced using outer class, eg Outer.Inner.blah(). |
| Operator Overloading | Defined as static methods, eg for a class Fraction to override +<br>public static Fraction operator+(Fraction lhs, Fraction rhs) {}<br>[Convention is to use lhs and rhs]<br>Not all languages in .NET will support operator overloading and thus will not use these methods – worth adding separate add() method.<br>Be careful – make use intuitive.<br>If overloading ==, must also overload !=, same with >, < etc.<br>Should also override Equals if overloading == |
| Conversion operators | Can overload how compiler will convert between types when casting<br>eg Fraction f = 1.67; myInt = (int)f;<br>use public static implicit operator Fraction (int theInt) { } |
| Structs | A simple user defined type, a lightweight alternative to a class. Can contain methods, properties etc. Doesn't support inheritance. Does support multiple interfaces. A struct is a value type.<br>Useful in arrays, but not in collections as boxing is required.<br>Define similar to class:<br>public struct Ade<br>{<br>  public SomeMethod()<br>  {}<br>}<br>Create using new operator (although do't have to!).<br>Ade x = new Ade(); |
| Interfaces | Short begine with I<br>public interface IAde:*baseclass*<br>{<br>  void Read();<br>  int Status{get; set; }<br>}<br>No access modifiers for methods/properties;<br>Interfaces can also implement other interfaces.<br><br>Can cast an object to the interface to use the interface, or use methods directly:<br>Document doc = new Document("ade");<br>IAde iaDoc = doc as IAde;<br>or IAde iaDoc = (IAde)doc;<br>iaDoc.Read();<br>doc.Read();<br><br>Can test interface using is:<br>if (doc is IAde) …<br>The as operator returns null (rather than an error) if cast fails.<br>In a class implementing the interface, can put interface name as part of method declaration. Useful if two interfaces have same method name. |

| | |
|---|---|
| | If explicit implmentaion then method is only visible when object is cast to the interface.<br>eg void IAde.Read() { }  [Note – no access modifier]<br>This allows implemented interface to be hidden if required. |
| Arrays | Arrays are objects and thus have a stack of methods available eg Copy, Sort, BinarySearch,<br>int [] myArray;<br>myArray = new int[5];  [First element is 0]<br>Array of value types are value types – not boxed objects.<br>Button[] myArray = new Button[3];<br>- does not create objects only null references – stll have create and assign button objects.<br>Access element using [index], eg myArray[3]<br>Multi-dimensional arrays, inc initiliastion<br>int [,] x = new int[2,2];<br>Initiliastion<br>int [,] x = { {1,2}, {3,4} }<br>Jagged arrays – an array of arrays<br>int [][] x<br>Arrays can be converted if type of the arrays can be converted |
| Params array | Can pass in multiple parameters to a method using a params array:<br>eg void MyMethod(params int[i] intVals)<br>MyMethod(1,2,3); or MyMethod(myIntArray); |
| Indexers | Allows access to a class as if it were an array.<br>Effectively overloads the [] operator.<br>Declare an indexer within a class as:<br>returnType this [accessType argument] { get; set;}<br>eg public string this[int i]<br>{  get { }<br>  set { }<br>}<br>then can access as obj[10] where obj is an instance of the class.<br>The accessType can be any type does not have to be an int.  Can also overload using different accessTypes (eg an int and a String indexer) |
| Collection Interfaces | Various interfaces that classes can supprt to provide collection functionality, eg:<br>IEnumerable: allows support of foreach<br>ICollection: provide copy, count etc<br>IComparer: allows collection sorting |
| Collection Types | Array: as above<br>ArrayList: A dynamicall sized array.  Use Add, Remove etc<br>Queue: fifo collection.  Use enqueue, dequeue, peek<br>Stack: lifo collection. Use pop, push, peek |
| Dictionaries | Associates values with a key.  Any kind of object can be associated with any type of key.<br>Hashtable: Ietm, Add, Contains, Remove<br>IDictionary: interface to implement |
| Strings | string x = "abc"; or string x = @"abc"; # says treat string literally (ignore escape characters)<br>string intStr = myInt.ToString();<br>Available methods: eg Copy, Compare, Format, SubString, ToUpper, Trim, Split (breaks into substrings)<br>StringBuilder class for dynamically building and processing strings.  Methods available: Append, Insert, Remove, Replace. |
| Exception Handling | throw new System.Exception("…");<br>try { }<br>catch { } or catch (Exception type) { }<br>finally { }<br>System.Exception provides Message(), StackTrace(), InnerException(), TargetSite()<br>Custom exceptions must derive from System.ApplicationException<br>InnerException allows Exception to be saved as part of throwing a new |

| | |
|---|---|
| | Exception – these can be nested.<br>Rethrowing exceptions: throw; or throw Exception; |
| Delegates and Events | A delegate is a reference type used to encapsulate a method with a particular signature.<br>public delegate int MyDelegate(params);<br>Declare a method that uses delegate<br>public void Sort(MyDelegate delFunc)<br>{ x = delFunc(params);<br>} |
| XML Documentation | Generate XML doc from code using /doc compiler switch<br>Reads comments marked with ///<br>Use tags such as <summary>, <returns>, <param><br>eg ///<summary>This class does this<summary> |