C# Language Reference

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

Ore constraints	Locignment (=)
Operators	Assignment (=) Arithmetic (+, -, *, /, %(modulus))
	Increment (++, +=, +* etc), decrement ()
	Relational (==, !=, >, >=, < , <=)
	Conditional(&&, , ! -> Note C# will short circuit expressions)
	Logical (&, ^,) Ternary (cond-expr ? expr1 : expr2)
	reinary (cond expr.: expr.: expr.)
Preprocessor	#define, #if etc
	<pre>#region name ->#endregion - marks a block of collapsible code</pre>
Classes	<pre>public class Ade:base-class { }</pre>
	Ade a = new Ade();
Access Modifiers	public : no restrictions
	private : only accessible to class protected : only accessible to class and subclasses
	internal : accessible to any class in assembly
	protected internal : == protected or internal
Methods	<pre>return-type Name(params) { }</pre>
Constructors	Same name as class and no return type. Can have multiple constructors with
	different param lists.
	Copy constructor must be created manually by passing an object in to a constructor method.
	A static constructor will run before any instance of the class is created
Destructor	Should only be used if there are unmanaged resources
	Called by garbage collector ~ClassName()
Dispose	Can define a Dispose method - implement interface IDisposable.
	Should suppress GC using GC.SuppressFinalize(this); Called automaticalling in using clauses eg
	using (x = new XYZ())
	{ } Dispose called automatically.
	this is the current object
Within class reference	base is the super class object
Static members	Belong to and referenced by the class name
	Cannot be referenced using an object instance
Params – by	Default is by value for value types
	Use (ref int x) to pass by reference
reference/by value	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
Overloading methods	work.
:	
Properties	Make instance variables private - access is via properties public int Xyz
	{ get { return Xyz; }
	<pre>set { Xyz = value } }</pre>
	get or set are optional
	Can then use property as if it were a normal variable. eq a.Xyz++
Inheritance	To override a base class method
	base class must define method as virtual
	public virtual void open() to override it in child class
	public override void open()
	All methods are final by default.
	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 eg public new virtual Xyz() Use sealed keyword to make a class final so it can't be inherited
Abstract class/method	<pre>abstract public void Add(); Must be overriden by sub class. Base class must also be abstract abstract public class AdeBase { }</pre>
System.Object	<pre>Provides Equals(), GetHashCode(), GetType(), ToString(), Finalize(), MemberwiseClone(), ReferenceEquals()</pre>
Boxing/Unboxing	Boxing converts a value type to a reference type and is automatic eg int I =123; i.ToString(); Unboxing converts from object to a value type - must be explicit 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	<pre>Defined as static methods, eg for a class Fraction to override + public static Fraction operator+(Fraction lhs, Fraction rhs) {} [Convention is to use lhs and rhs] Not all languages in .NET will support operator overloading and thus will not use these methods - worth adding separate add() method. Be careful - make use intuitive. If overloading ==, must also overload !=, same with >, < etc. Should also override Equals if overloading ==</pre>
Conversion operators	Can overload how compiler will convert between types when casting eg Fraction f = 1.67; myInt = (int)f; use public static implicit operator Fraction (int theInt) { }
Structs	<pre>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. Useful in arrays, but not in collections as boxing is required. Define similar to class: public struct Ade { public SomeMethod() {} } Create using new operator (although do't have to!). Ade x = new Ade();</pre>
Interfaces	<pre>Short begine with I public interface IAde:baseclass { void Read(); int Status{get; set; } } No access modifiers for methods/properties; Interfaces can also implement other interfaces. Can cast an object to the interface to use the interface, or use methods directly: Document doc = new Document("ade"); IAde iaDoc = doc as IAde; or IAde iaDoc = (IAde)doc; iaDoc.Read(); doc.Read(); Can test interface using is: if (doc is IAde) The as operator returns null (rather than an error) if cast fails. In a class implementing the interface, can put interface name as part of method declaration. Useful if two interfaces have same method name.</pre>

Tf explicit implementation then method is only visible when object is cast to the interface. eg void IAde.Read() () [Note - no access modifier] This allows implemented interface to behiden if required. Arrays Arrays are objects and thus have a stack of methods available eg Copy, Sort, sinarysearch, int [] myArray - new int[b]; [Pisk element is 0] Array of value types are value types - not boxed objects. Builton] [spiray - new int[b]; Dutton objects. Access element, using [index], eg myArray[] Multi-dimensional arrays, inc: initilisation int, [] x - end array of arrays 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: eg void MyMethod(params int[]) [Intyl] inficities (] construction (] operator. beclars an indexe within a class as: returnfyme this [accessType argument] { ept set;} eg public string this[int [] i ent		TE suplicit implementation than method is set. """"
eg vois inde.mead() () [note - no access modifier] Arrays Arrays are objects and thus have a stack of methods available eg Copy, Sort, Enargysearch, in [] myEaray; Arrays Bardy Arrays Arrays Arrays Arrays Arrays Arrays Arrays Arrays Arrays Bardy Arrays Bardy Arrays Arrays <tr< th=""><th></th><th></th></tr<>		
This allows implemented interface to be hidden if required. Arrays Arrays are objects and thus have a stack of methods available eg Copy, Sort, int II supress, int II maydray, int II maydray, int II maydray, of value types are subin types are not howed objects. Nutlon[] makray - new Rutton[3]; - does not create objects only null references - still have create and assign button objects. Nutlon[] makray - new Rutton[3]; - does not create objects only null references - still have create and assign button objects. Nutlon[] makray = new Rutton[3]; - does not create objects only null references - still have create and assign button objects. Nucleoses element using [index], eq myArray[3] Multi-dimensional arrays, inc initiliation int [1] x = twint[2,2]; int [1] y = twint[2,2]; int [1] y = twint[2,2]; int [1] y = twint[2,2]; int [1] y = twint[2,2]; int create an indexer within a class as: returnype this[] (creating argument] (get; set;) eg public scring this[] twint[] (get []] i the can access as obj[10] where obj is an instance of the class. The accessfype can be any type does not have to be an int. Can also overload using different accessfype (get an int a string indexer) Collection Interfaces Variause interfaces that classes can supprt to provide collection thromerable: allows oblection sorting Collection Types Nray: as above the class into collection. We enough, many peet theorement interfaces that a kay, May lind of object can be essociated with any type of ky. Nashidable: they, Add, Contains, Reeve Dictionaries Nray: as above enderication; stri		
Arrays are objects and thus have a stack of methods available eg Copy, Sort, sinarysysarch, mytray - new int(5)/ [First element is 0] Array of view types are value types - not boxed objects. By deal not create objects only null references - stil have create and assign button objects. Access element using [index], eg myArray(3] Mull-edemonical arrays, inc initiliastion int (1) x - new int(2,2); initiliastions arrays, inc initiliastion int (1) x - (1,2, (3,4)) Thir(1) x - (1,2, (3,4)) arrays can be converted if type of the arrays can be converted Arrays can be converted if type of the arrays can be converted Arrays can be converted if type of the arrays can be converted Arrays can be converted if type of the arrays can be converted Arrays can be converted if type of the arrays can be converted Arrays can be converted if type of the arrays. Trifectively overlaads the (1) operator: returnType this [accessType argument] (get: set;) eg ublic arring this[int 1] (get { }		
Binarysearch,' Binarysearch,' myArray = new int[b]; [First element is 0] Array of value types are value types = not boxed objects. Button[] myArray = new int[b]; [First element is 0] Array of value types are value types = not boxed objects. Button[] myArray = new int[b]; [First element is 0] Array of value types are value types = not boxed objects. Button[] myArray = new int[b]; [First element is 0] Array of value types are value types = not boxed objects. Button[] myArray = new int[b]; [First element is 0] Array of value types are value types = not boxed objects. Button[] myArray = new int[b]; [First element is 0] Array of value types are value types = not boxed objects. Button[] myArray = new int[b]; [First element is 0] Array of value types are value types = not boxed objects. Button[] value types argument is 0] Arrays can be converted if type of the arrays can be converted Params array Can pass in multiple parameters to a method using a parama array: eq value whethod (parama int[1] intvals) MyMethod(1, 2, 3); or MyMethod(myThAtray); Into a pass in multiple parameters to a method using a parama array: eq value whethod (parama int[1] intvals) MyMethod(1, 2, 3); or MyMethod(myThAtray); Intervent arraysearray (parama int[1] intvals) Params array Can pass in multiple parameters to a method using arraysearraysearray and be converted Indexers A		
BinarySearch, int II myArray: new int(5); [First element is 0] hyArray: mex int(5); myArray: area interview evalue grass - not boxed objects. hyperstand interview objects cnly mult references - stil have create and assign button objects. Access element using [Index], eq myArray(3) Multi-dimensional arrays, inc initiliaistion int [] x - ew int(2,2); int [] x - ew int(2,2); int(2,3); or MyMethod (myIntArray); Indexers Allows access to a class as if it were an array. Effectively overloads the [] operator. Declare an indexer within a class as: the accessitype argument] [] get {] }	Arravs	Arrays are objects and thus have a stack of methods available eg Copy, Sort,
myArtry - new int(5); [lists element is 0] Array of value types are value types - not based objects. Button[] myArray = new Button[3]; - does not create objects only null references - stil have create and assign button objects. Access silement using (index], eg myArray[3] Multi-dimensional arrays; ino intiliastion Inttiliaries Jaged arrays - an array of arrays int [1] x 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: eq void MyMethod(gorams int[1] intVals) MyMethod(1,2,3); or MyMethod(myIntArray); Indexers Allows access to a class as if is ware an array. Effectively overloads the [] operator. beclare an indexer within a class as: returnType this [accessType argument] (get; set;) eg public string this[int1] (at 2 at {}) the accessrupe can be any type does not have to be an int. Can also overload using different accessrupes (a guant the add atting indexer) Collection Interfaces Various interfaces the appet to provide collection interfaces the accessrupe calle aby only for foreach Icollection: growide copy, count etc Icolle		
Array of value types are value types - not boxed objects. Button[] mykray - new Nutton[]] - des not create objects only null references - stil have create and assign button objects. Access element using [index], eg mykray[3] Multi-dimensional array, inn initiliation int [] x = new int[2,2]; Init[] x = new int[] new int[] Indexers Allows access to a class as if it were an array. #fective] versionad the [] operator. Declement accessinge on be any type does not have to be an int. Can also overload using different accessinges (eg an int and a String index)		
Button[] myArray ¹ - new Nutton[3]; - dees not create objects. Access of create objects. Access of create objects. Access of create objects. Nutri-dimensional arrays, inc initiliaation int [,] x = new int[2,2]; Instiliaation int [,] x = { [1,2], (3,4] } Arrays can be converted if type of the arrays can be converted Params array Can pass in multiple parameters to a method using a parame array: eg wold MyMethod(parama int[1] intvals) MyMethod(1,2,3); or MyMethod(myIntArray); Indexers Allows access to a class as if it were an array. Eclare an indexer within a class as: returnType this [accessType argument] [get; set;) eg public aindexer within a class as: returnType this [accessType argument] [get; set;) eg public andexer within a class as: 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, ed; IEnumerable; allows support of foreach ICollection sprovide copy, count etc Toompare; allows collection sorting Collection Types Array: as above Array: as above Array		
- does not create objects only null references - stil have create and assign button objects. Access element using [Index], eg myArray[3] Multi-dimensional arrays, inc initiliation int [,] x = new int[2,2]; Initiliation int [,] x - { (1,2), (3,4) } Jagged arrays - an array of arrays int [U[] x 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; eg void MyMethod(J,2,3); or MyMethod(J,2,3); Mikrea access to a class as if it were an array. Effectively vorelows the [] operatively Mikrea access to a class as if it were an array. Effectively vorelows the [] operatively Return indexer within a loss store returning this [int i] (] got [] (] got [] if extra access as obj[10] where obj is an instance of the class. The accessType can be any type does not have to be an int. Can also overload using different accessType any using other obj. Collection Interfaces Various interfaces that classes can supprt to provide collection functionality, eg: Innucrionality, eg: Innucrion		
button objects. Access element using [index], eg myArray[3] Multi-dimensional arrays, inc initiliation int [,] x = new int[2,2]; Initiliation int [,] x = new int[2,1]; Initiliation int [,] x = [(1,2], (3,4) } Jagged arrays - an array of arrays Int [,] x = [(1,2], (3,4) } Jagged arrays - an array of arrays Int [,] x = [(1,2], (3,4) } Jagged arrays - an array of arrays Int [,] x = [(1,2], (3,4) } Jagged arrays - an array of arrays Int [,] x = [(1,2], (3,4) } Jagged arrays - an array of arrays 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: eg vobic string thin a class as: returntype this [accesarype argument] (get; set;) eg public string this[int i] i (get { } j set { } <		
Access element using [index], eg myArray[3] Multi-dimesional arrays, inc initiliation int [,] x = new inl[2,2]; intiliation int [,] x = ((1,2), (3,4)) jacged arrays - an array of arrays int [][] x 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: eg void MyMethod(params int[1] intvals] MyMethod(12,3); or MyMethod(myIntray); Indexers Allows access to a class as If it were an array. Effectively overloads the [] operator. Declare an indexer within a class as as; republic string docessrype argument] (get; set;) eg voit [] i the can access as obj[10] where obj is an instance of the class. The accessrype can be any type does not have to be an int. Can also overload using different accessrypes (eg an int and a String indexer) Collection Interfaces Various interfaces that classes can supprt to provide collection functionality, eg: IEnumerable: allows support of foreach ICollection: provide copy, count etc ICollection sprovide copy, count etc ICollection. Use pop, push, peek Dictionaries Array: as above Array: as above Array: istack: 11f collection. Use enqueue, dequeue, peek Stack: 11f collection. Use pop, push, peek Strings Arsoclates values with a key. Any kind of object can be associated with any type of key. Hashtable: Tetn, Add, Contains, Remove IDictionary: interface to implement String interface to implement istring interface to implement Available: Rethox supering) Strings Etception Handling throw new System.Exception (""); try {} catch {} or catch (Exception type) {} finally {} System.Exception provides Message(), StackTrace(), ImerException(), <th></th> <th></th>		
Wull:-dimensional airays, inc initiliation int [,] x = new init[,2]; Init[]] x 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: eq void Mydethod(parama int[]) intVals) Mydethod(nama int[]) x 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: eq void Mydethod(myIntArray); Indexers Allows access to a class as if it were an array. Effectively overloads the [] operator. Declare an indexer within a class as: return?py this [access?pe acument] (get; set;) eq public string this[int i] { get { } } the accessType can be any type does not have to be an int. Can also overload using different accessTypes (eq an int and a String Indexer) Collection Interfaces Various interfaces that classes can supprt to provide collection functionality, eg; Immwrahle: allows oupport of foreach icollection: provide copy, count etc Icollection is provide copy, count etc Icollection orring Collection Types Array: as above Array: as above Array: as above Array: set boy (Array). Katatable: letn, Add, Contains, Remove Ibictionaries Associates values with a key. Any kind of object can be associated with any type of key. Hastable: letn, Add, Contains, Remove Ibictionary: interface of arrays = #abc"; or string x = #abc"; format, SubString, ToUpper, Trin, Split (breaks ind substrings) String Nift = myInt.ToString(); Available methods: eg Copy, Compare, Format, SubString, ToUpper, Trin, Split (breaks ind substrings) S		
int [,] x = new int[2,2]; Initiliation int [,] x = { (1,2), (3,4) } Jagged arrays - an array of arrays int [] [] x 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: eg void MyMethod(params int[] intVals) MyMethod(1,2,3); or MyMethod(myIntArray); Indexers Allows access to a class as if it were an array. Effectively overloads the [] operator. Declare an indexer within a class as: returnType this [accessType argument] (get; set;) eg public string this[int i] (get [] set []); then can access as obj[10] where obj is an instance of the class. The accessTypes 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: [Enumerable: allows support of foreach [Collection: provide copy, count etc [Comparer: allows collection. Use enguee, dequeue, peek Stack: life collection. Use enguee, dequeue, peek Stack: life collection. Use pop, push, peek Dictionaries Associates values with a key. Any kind of object can be associated with any type of key. Rashtable: Tetm, Add, Contains, Remove Exception Handling string x = "abc"; or string x = 8"abc"; 4 says treat string literally (ignore escape characters) string litts r = mytht.Tottring(); Available methods: eg Copy, Compare, Format, Substring, Toupper, Trim, Split (breaks ind substrings) StringSuiter engue. Remove, Replace. thow ne		
int [,] x = { (1,2), (3,4) } Jagged arrays - an array of arrays Arrays can be converted if type of the arrays can be converted Params array Can pass in multiple parameters to a method using a parame array: eg void MyMethod(parame int[] intvals) MyMethod(1,2,3); or MyMethod(myIntArray); Indexers Allows access to a class as if it were an array. Effectively overloads the [] operator. Declare an indexor within a class as: returnType this [accessType argument] { get; set; } eg public atring this[int i] { get { } set { } } (add) (Declare an access as obj[10] where obj is an instance of the class. 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: Innumerable: allows support of foreach ICollection: provide copy, count etc IComparer: allows collection sorting Collection Types Array: as above ArrayList: A dynamicall sized array. Use Add, Remove etc Queue: firs collection. Use enqueue, dequeue, peek Stack: life collection. Use pop, push, peek Dictionaries Associates values with a key. Any kind of object can be associated with any type of key. Reshtable: Tetm, Add, Contains, Remove IDictionary: interface to implement Strings string x = "abc"; or string x = @"abc"; 4 says treat string literally (ignore eccape characters) string infst = myInt.ToString(); Available methods: eg Copy, Compare, Format, SubString, ToUpper, Trim, Split (breaks ind substrings) StringNuilder class for dynamically building and processing str		
Jagged arrays - an array of arrays int [[] x 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: eq void Wykethod(nams int[1] intvals) Mykethod(1,2,3); or Mykethod(myIntArray); Indexers Allows access to a class as if it were an array. Effectively overloads the [] operator. Declare an indexer within a class as: returnType this [accessType argument] (get; set;) eg public string this[int i] { et (} } then can access as obj[10] where obj is an instance of the class. 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: IBnumerable: allows coupport of foreach iCollection: provide copy, count etc iComparer: allows collection sorting Collection Types Array: as above ArrayLis: A dynamical] sized array. Use Add, Remove etc Cueue: filo collection. Use enqueue, dequeue, peek Stack: life collection. Use enqueue, dequeue, peek Stack: life collection. Use pop, push, peek Dictionaries Associates values with a key. Any kind of object can be associated with any type of key. Hashtable: Item, Add, Contains, Remove IDictionary: Interface to implement Strings atring x = "abc"; or string x = @"abc"; # says treat string literally (ignore escape characters) string litstr = myInt.ToString(); Available methods: eg Copy, Compare, Format, Substring, ToUpper, Trin, Split (breaks into substring) StringBuilder class for dynamically building and processing strings. Methods available: Append, Insert, Remove, Replace. <t< td=""><td></td><td>Initiliastion</td></t<>		Initiliastion
int [[] x 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: eg void MyMethod(params int(1) intVals) MyMethod(1,2,3); or MyMethod(myIntArray); Indexers Allows access to a class as if it were an array. Effectively overloads the [] operator. beclare an indexer within a class as: returnType this [accessType argument] (get; set;) eg public string this[int i] i get {} set {} i then can access as obj[10] where obj is an instance of the class. 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: Incollection: provide copy, count etc ICollection Types Array: as above Dictionaries Dictionaries Associates values with a key. Any kind of object can be associated with any type of kay. Hasthabic: letm, Add, Contains, Remove D		int [,] x = { {1,2}, {3,4} }
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: eq void WyMethod(params int[1] intvals) HyMethod(1,2,3); or MyMethod(myIntArray); Indexers Allows access to a class as if it were an array. Effectively overloads the [] operator. Declare an indexer within a class as: returnType this [accessType argument] (get; set;) eg public string this[int i] ; et [] ; et [] Collection Interfaces Various interfaces that classes can supprt to provide collection functionality, eg; Ennuerable: allows support of foreach ECollection: provide copy, count etc ECollection: provide copy, count etc ECollection: Types Array: as above ArrayLas: A dynamical] sized array. Use Add, Remove etc Cueue: filo collection. Use enqueue, dequeue, peek 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. Hashtable: Ietm Add, Contains, Remove EDictionary interface to implement Strings string x = "abc"; or string x = @"abc"; # says treat string literally (ignore escage characters) string liftsr = myInt.ToString(); Available methods: eg Copy, Compare, Format, SubString, ToUpper, Trin, Split (breaks into substring) StringBuilder class for dynamically building and processing strings. Methods available: Append, Inset, Remove, Replace. Exception Handling throw new System.Exception ("_"); try { cath { or cath (Exception type) { } finally { } System.Exception provides Message(), StackTrace(), InnerException(),		Jagged arrays - an array of arrays
Params array Can pass in multiple parameters to a method using a parame array: ey void NyMethod(parame int[1] intVals) MyMethod(1,2,3); or MyMethod(myIntArray); Indexers Allows access to a class as if it were an array. Effectively overloads the [] operator. Declare an indexer within a class as: returnType this [accessType argument] { get; set;} eg public string this[int i] { get { } set { } } then can access as obj[10] where obj is an instance of the class. 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: Innumerable: allows support of foreach IcOllection: provide copy, count etc IComparer: allows collection sorting Collection Types Array: as above Array: as above Array: as above Array: interfaces that equeue, dequeue, peek Stack: life collection. Use enqueue, dequeue, peek Stack: life collection. Use enqueue, dequeue, peek Stack: life collection. Use pop, push, peek Dictionaries Associates values with a key. Any kind of object can be associated with any type of key, Hashtable: letm, Add, Contains, Remove Ibictionary: interface to implement Strings string x = "abc"; or string x = @"abc"; # says treat string literally (ignore escape characters) string intstr = myInt.ToString(); Available methods: eg Copy. Compare, Format, SubString, ToUpper, Trim, Split (breaks into substring) Stringbuilder class for dynamically building and processing strings. Methods available: append, Insert, Remove, Replace. Exception Handling throw new System.Exception(""); try { } cach { } or cach (Exception typ		
eq void Mydethod(barams int[1] intVals) Mydethod(1,2,3); or Mydethod(myIntArray); Allows access to a class as if it were an array. Effectively overloads the [] operator. Declare an indexer within a class as: returnType this [accessType argument] { get; set; } eq public string this[int i] (get { }) set { } } then can access as obj[10] where obj is an instance of the class. 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; Innumerable: allows support of foreach ICOllection: provide copy, count etc ICOmparer: allows collection sorting Collection Types Array: as above ArrayList: A dynamicall sized array. Use Add, Remove etc Queue: fifo collection. Use enqueue, dequeue, peek Stack: lifo collection. Use enqueue, dequeue, peek 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. Hashtable: letm, Add, Contains, Remove IDictionary: interface to implement Strings string untStr = myInt.ToString(); Available methods: eg Copy, Compare, Format, SubString, ToUpper, Trim, Split (breaks into substrings) StringVallder class for dynamically building and processing strings. Methods available: Append, Insert, Remove, Replace. Exception Handling throw new System.Exception(""); try { } catch { } or catch (Exception type) () finally { } System.Exception provides Message(), StackTrace(), InnerException(),		Arrays can be converted if type of the arrays can be converted
eq void Mydethod(barams int[1] intVals) Mydethod(1,2,3); or Mydethod(myIntArray); Allows access to a class as if it were an array. Effectively overloads the [] operator. Declare an indexer within a class as: returnType this [accessType argument] { get; set; } eq public string this[int i] (get { }) set { } } then can access as obj[10] where obj is an instance of the class. 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; Innumerable: allows support of foreach ICOllection: provide copy, count etc ICOmparer: allows collection sorting Collection Types Array: as above ArrayList: A dynamicall sized array. Use Add, Remove etc Queue: fifo collection. Use enqueue, dequeue, peek Stack: lifo collection. Use enqueue, dequeue, peek 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. Hashtable: letm, Add, Contains, Remove IDictionary: interface to implement Strings string untStr = myInt.ToString(); Available methods: eg Copy, Compare, Format, SubString, ToUpper, Trim, Split (breaks into substrings) StringVallder class for dynamically building and processing strings. Methods available: Append, Insert, Remove, Replace. Exception Handling throw new System.Exception(""); try { } catch { } or catch (Exception type) () finally { } System.Exception provides Message(), StackTrace(), InnerException(),		
NyMethod(1,2,3); or MyMethod(myIntArray); Indexers Allows access to a class as if it were an array. Brfactively overlads the [] operator. beclare an indexer within a class as: returnType this [accessType argument] { get; set; } eg public string this[int i] (get ()	Params array	
Indexers Allows access to a class as if it were an array. Effectively overloads the [] operator. Declare an indexer within a class as: return/pe this [accessType argument] { get; set;} eg public string this[int i] { get { } set { } } then can access as obj[10] where obj is an instance of the class. 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: IEnumerable: allows support of foreach ICollection: provide copy, count etc IComperer: allows collection. Use enqueue, dequeue, peek Stack: lifo collection. Use pop, push, peek Dictionaries Array: as above ArrayList: A dynamicall sized array. Use Add, Remove etc Queue: fifo collection. Use pop, push, peek Dictionaries Associates values with a key. Any kind of object can be associated with any type of key. Hashtable: letm, Add, Contains, Remove IDictionary: interface to implement Strings string x = "abc"; or string x = @"abc"; f says treat string literally (ignore escape characters) string luider class for dynamically building and processing strings. Methods available: Append, Insert, Remove, Replace. Exception Handling throw new System.Exception(""); try { } catch { } or catch (Exception type) { } finally { } System.Exception provides Message(), StackTrace(), InnerException(),		
InductorEffectively overloads the [] operator.Declare an indexer within a class as: returnType this [accessType argument] { get; set;} eg public string this[int i] { get { } set { } }(get { } set { } }Collection InterfacesVarious interfaces that classes can supprt to provide collection functionality, eg: IEnumerable: allows support of foreach ICollection: provide copy, count etc ICollection: provide collection. Use pop, push, peekCollection TypesArray: as above ArrayList: A dynamicall sized array. Use Add, Remove etc Queue: fifo collection. Use pop, push, peekDictionariesAssociates values with a key. Any kind of object can be associated with any type of key. Hashtable: IeEm, Add, Contains, Remove IDictionary: interfacet of implementStringsstring x = "abc"; or string x = @"abc"; # says treat string literally (ignore escape characters) Stringsulter class of dynamically building and processing strings. Methods available methods: eg Copy, Compare, Format, SubString, ToUpper, Trim, Split (preaks into substring;) Stringsulter class of dynamically building and processing strings. Methods available: Append, Insert, Remove, Replace.Exception Handlingthrow new System.Exception(""); try (} catch { } or catch (Exception type) { } finally (} System.Exception provides Message(), StackTrace(), InnerException(),		MyMethod(1,2,3), of MyMethod(MyThtAffay),
InductorEffectively overloads the [] operator.Declare an indexer within a class as: returnType this [accessType argument] { get; set;} eg public string this[int i] { get { } set { } }(get { } set { } }Collection InterfacesVarious interfaces that classes can supprt to provide collection functionality, eg: IEnumerable: allows support of foreach ICollection: provide copy, count etc ICollection: provide collection. Use pop, push, peekCollection TypesArray: as above ArrayList: A dynamicall sized array. Use Add, Remove etc Queue: fifo collection. Use pop, push, peekDictionariesAssociates values with a key. Any kind of object can be associated with any type of key. Hashtable: IeEm, Add, Contains, Remove IDictionary: interfacet of implementStringsstring x = "abc"; or string x = @"abc"; # says treat string literally (ignore escape characters) Stringsulter class of dynamically building and processing strings. Methods available methods: eg Copy, Compare, Format, SubString, ToUpper, Trim, Split (preaks into substring;) Stringsulter class of dynamically building and processing strings. Methods available: Append, Insert, Remove, Replace.Exception Handlingthrow new System.Exception(""); try (} catch { } or catch (Exception type) { } finally (} System.Exception provides Message(), StackTrace(), InnerException(),	Indovors	Allows access to a class as if it were an array.
Declare an indexer within a class as: returnType this [accessType argument] { get; set;} eg public string this[int i] { get { } set { } } then can access as obj[10] where obj is an instance of the class. 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 InterfacesVarious interfaces that classes can supprt to provide collection functionality, eg: IBnumerable: allows support of foreach ICollection: provide copy, count etc IComparer: allows collection. Use enqueue, dequeue, peek Stack: lifo collection. Use enqueue, dequeue, peek Stack: lifo collection. Use pop, push, peekDictionariesAssociates values with a key. Any kind of object can be associated with any type of key. Hashtable: Term, Add, Contains, Remove IDictionary: interface to implementStringsstring x = "abc"; or string x = @"abc"; # says treat string literally (ignore escape characters) string intStr = myInt.ToString(); Available methods: eg Copy, Compare, Format, SubString, ToUpper, Trim, Split (breaks into substrings) StringBuilder class for dynamically building and processing strings. Methods available: Append, Insert, Remove, Replace.Exception Handlingthry { } catch { } or catch (Exception type) { } finally { } System.Exception provides Message(), StackTrace(), InnerException(),	Indexers	-
eg public string this[int i] (get { } set { } set { } ; set { } ; then can access as obj[10] where obj is an instance of the class. 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 InterfacesVarious interfaces that classes can supprt to provide collection functionality, eg: IEnumerable: allows support of foreach ICollection: provide collection sortingCollection TypesArray: as above ArrayList: A dynamicall sized array. Use Add, Remove etc Queue: fifo collection. Use popue, peek Stack: lifo collection in prime associated with any type of key. Hashtable: Ietm, Add, Contains, Remove IDictionary: interface to implementStringsstring x = "abc"; or string x = @"abc"; # says treat string literally (ignore escape characters) StringBuilder class for dynamically building and processing strings. Methods available: Append, Insert, Remove, Replace.Exception Handlingthrow new System.Exception (""); try { } catch { } or catch (Exception type) { } finally { } System.Exception provides Message(), StackTrace(), InneException(),		
<pre>(</pre>		returnType this [accessType argument] { get; set;}
set () } then can access as obj[10] where obj is an instance of the class. 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: IEnumerable: allows support of foreach ICollection: provide copy, count etc IComparer: allows collection sorting Collection Types Array: as above ArrayList: A dynamicall sized array. Use Add, Remove etc Queue: fifo collection. Use enqueue, dequeue, peek 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. Hashtable: Ietm, Add, Contains, Remove IDictionary: interface to implement Strings string x = "abc"; or string x = @"abc"; # says treat string literally (ignore escape characters) string intStr = myInt.ToString(); Available methods: eg Copy, Compare, Format, SubString, ToUpper, Trim, Split (breaks into substrings) StringBuilder class for dynamically building and processing strings. Methods available: Append, Insert, Remove, Replace. Exception Handling throw new System.Exception(""); try { } catch () or catch (Exception type) { } finally { } System.Exception provides Message(), StackTrace(), InnerException(),		eg public string this[int i]
<pre>} then can access as obj[10] where obj is an instance of the class. 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: IRnumerable: allows support of foreach ICollection: provide copy, count etc IComparer: allows collection sorting Collection Types Array: as above ArrayList: A dynamicall sized array. Use Add, Remove etc Queue: fifo collection. Use enqueue, dequeue, peek 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. Hashtable: Ietm, Add, Contains, Remove IDictionary: interface to implement Strings string x = "abc"; or string x = @"abc"; # says treat string literally (ignore escape characters) string intStr = myInt.ToString(); Available methods: eg Copy, Compare, Format, SubString, ToUpper, Trim, Split (breaks into substrings) StringBuilder class for dynamically building and processing strings. Methods available: Append, Insert, Remove, Replace. Exception Handling throw new System.Exception(""); ty { } catch { } or catch (Exception type) { } finally { } System.Exception provides Message(), StackTrace(), InnerException(), </pre>		{ get { }
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 InterfacesVarious interfaces that classes can supprt to provide collection functionality, eg: IEnumerable: allows support of foreach ICollection: provide copy, count etc IComparer: allows collection sortingCollection TypesArray: as above ArrayList: A dynamicall sized array. Use Add, Remove etc Queue: fifo collection. Use enqueue, dequeue, peek Stack: lifo collection. Use pop, push, peekDictionariesAssociates values with a key. Any kind of object can be associated with any type of key. Hashtable: Ietm, Add, Contains, Remove IDictionary: interface to implementStringsstring x = "abc"; or string x = @"abc"; # says treat string literally (ignore escape characters) string intStr = myInt.ToString(); Available methods: eg Copy, Compare, Format, SubString, ToUpper, Trim, Split (breaks into substrings) StringBuilder class for dynamically building and processing strings. Methods available: Append, Insert, Remove, Replace.Exception Handlingthrow new System.Exception(""); try { } catch { } or catch (Exception type) { } finally { } System.Exception provides Message(), StackTrace(), InnerException(),		set { }
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 InterfacesVarious interfaces that classes can supprt to provide collection functionality, eg: IEnumerable: allows support of foreach ICollection: provide copy, count etc IComparer: allows collection sortingCollection TypesArray: as above ArrayList: A dynamicall sized array. Use Add, Remove etc Queue: fifo collection. Use enqueue, dequeue, peek Stack: lifo collection. Use pop, push, peekDictionariesAssociates values with a key. Any kind of object can be associated with any type of key. Hashtable: Ietm, Add, Contains, Remove IDictionary: interface to implementStringsstring x = "abc"; or string x = @"abc"; # says treat string literally (ignore escape characters) string intStr = myInt.ToString(); Available methods: eg Copy, Compare, Format, SubString, ToUpper, Trim, Split (breaks into substrings) StringBuilder class for dynamically building and processing strings. Methods available: Append, Insert, Remove, Replace.Exception Handlingthrow new System.Exception(""); try { } catch { } or catch (Exception type) { } finally { } System.Exception provides Message(), StackTrace(), InnerException(),		
using different accessTypes (eg an int and a String indexer)Collection InterfacesVarious interfaces that classes can supprt to provide collection functionality, eg: IEnumerable: allows support of foreach ICollection: provide copy, count etc IComparer: allows collection sortingCollection TypesArray: as above ArrayList: A dynamicall sized array. Use Add, Remove etc Queue: fifo collection. Use enqueue, dequeue, peek Stack: lifo collection. Use pop, push, peekDictionariesAssociates values with a key. Any kind of object can be associated with any type of key. Hashtable: Ietm, Add, Contains, Remove IDictionary: interface to implementStringsstring x = "abc"; or string x = @"abc"; # says treat string literally (ignore escape characters) string intStr = myInt.ToString(); Available methods: eg Copy, Compare, Format, SubString, ToUpper, Trim, Split (breaks into substrings) StringBuilder class for dynamically building and processing strings. Methods available: Append, Insert, Remove, Replace.Exception Handlingthrow new System.Exception (""); try { } catch { } or catch (Exception type) { } finally { } System.Exception provides Message(), StackTrace(), InnerException(),		
Collection Interfaces Various interfaces that classes can supprt to provide collection functionality, eg: IEnumerable: allows support of foreach ICollection: provide copy, count etc IComparer: allows collection sorting Collection Types Array: as above ArrayList: A dynamicall sized array. Use Add, Remove etc Queue: fifo collection. Use enqueue, dequeue, peek 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. Hashtable: Ietm, Add, Contains, Remove IDictionary: interface to implement Strings string x = "abc"; or string x = @"abc"; # says treat string literally (ignore escape characters) string intStr = myInt.ToString(); Available methods: eg Copy, Compare, Format, SubString, ToUpper, Trim, Split (breaks into substrings) StringBuilder class for dynamically building and processing strings. Methods available: Append, Insert, Remove, Replace. Exception Handling throw new System.Exception (""); try { } catch { } or catch (Exception type) { } finally { } System.Exception provides Message(), StackTrace(), InnerException(),		
Functionality, eg: IEnumerable: allows support of foreach ICollection: provide copy, count etc ICollection: provide copy, count etc ICollection: provide copy, count etc ICollection: Display Array: as above ArrayList: A dynamicall sized array. Use Add, Remove etc Queue: fifo collection. Use enqueue, dequeue, peek 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. Hashtable: Ietm, Add, Contains, Remove IDictionary: interface to implement Strings string x = "abc"; or string x = @"abc"; # says treat string literally (ignore escape characters) string intStr = myInt.ToString(); Available methods: eg Copy, Compare, Format, SubString, ToUpper, Trim, Split (breaks into substrings) StringBuilder class for dynamically building and processing strings. Methods available: Append, Insert, Remove, Replace. trow new System.Exception(""); try { } catch { } or catch (Exception type) { } finally { } System.Exception provides Message(), StackTrace(), InnerException(),		using different accessiypes (eg an fit and a string indexer)
Functionality, eg: IEnumerable: allows support of foreach ICollection: provide copy, count etc ICollection: provide copy, count etc ICollection: provide copy, count etc ICollection: Display Array: as above ArrayList: A dynamicall sized array. Use Add, Remove etc Queue: fifo collection. Use enqueue, dequeue, peek 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. Hashtable: Ietm, Add, Contains, Remove IDictionary: interface to implement Strings string x = "abc"; or string x = @"abc"; # says treat string literally (ignore escape characters) string intStr = myInt.ToString(); Available methods: eg Copy, Compare, Format, SubString, ToUpper, Trim, Split (breaks into substrings) StringBuilder class for dynamically building and processing strings. Methods available: Append, Insert, Remove, Replace. trow new System.Exception(""); try { } catch { } or catch (Exception type) { } finally { } System.Exception provides Message(), StackTrace(), InnerException(),	Collection Interfaces	Various interfaces that classes can supprt to provide collection
IEnumerable: allows support of foreach ICollection: provide copy, count etc IComparer: allows collection sortingCollection TypesArray: as above ArrayList: A dynamicall sized array. Use Add, Remove etc Queue: fifo collection. Use enqueue, dequeue, peek Stack: lifo collection. Use pop, push, peekDictionariesAssociates values with a key. Any kind of object can be associated with any type of key. Hashtable: Ietm, Add, Contains, Remove IDictionary: interface to implementStringsstring x = "abc"; or string x = @"abc"; # says treat string literally (ignore escape characters) string intStr = myInt.ToString(); Available methods: eg Copy, Compare, Format, SubString, ToUpper, Trim, Split (breaks into substrings) StringBuilder class for dynamically building and processing strings. Methods available: Append, Insert, Remove, Replace.Exception Handlingthrow new System.Exception(""); try { } catch { } or catch (Exception type) { } finally { } System.Exception provides Message(), StackTrace(), InnerException(),	Collection Interfaces	
ICollection: provide copy, count etc IComparer: allows collection sortingCollection TypesArray: as above ArrayList: A dynamicall sized array. Use Add, Remove etc Queue: fifo collection. Use enqueue, dequeue, peek Stack: lifo collection. Use pop, push, peekDictionariesAssociates values with a key. Any kind of object can be associated with any type of key. Hashtable: Ietm, Add, Contains, Remove IDictionary: interface to implementStringsstring x = "abc"; or string x = @"abc"; # says treat string literally (ignore escape characters) string intStr = myInt.ToString(); Available methods: eg Copy, Compare, Format, SubString, ToUpper, Trim, Split (breaks into substrings) Stringsuider class for dynamically building and processing strings. Methods available: Append, Insert, Remove, Replace.Exception Handlingthrow new System.Exception(""); try { } catch { } or catch (Exception type) { } finally { } System.Exception provides Message(), StackTrace(), InnerException(),		
Collection Types Array: as above ArrayList: A dynamicall sized array. Use Add, Remove etc Queue: fifo collection. Use enqueue, dequeue, peek Stack: lifo collection. Use pop, push, peek Associates values with a key. Any kind of object can be associated with any type of key. Hashtable: Ietm, Add, Contains, Remove IDictionaries Strings string x = "abc"; or string x = @"abc"; # says treat string literally (ignore escape characters) string intStr = myInt.ToString(); Available methods: eg Copy, Compare, Format, SubString, ToUpper, Trim, Split (breaks into substrings) StringBuilder class for dynamically building and processing strings. Methods available: Append, Insert, Remove, Replace. Exception Handling throw new System.Exception (""); try { } catch { } or catch (Exception type) { } finally { } System.Exception provides Message(), StackTrace(), InnerException(),		
ArrayList: A dynamicall sized array. Use Add, Remove etc Queue: fifo collection. Use enqueue, dequeue, peek Stack: lifo collection. Use pop, push, peekDictionariesAssociates values with a key. Any kind of object can be associated with any type of key. Hashtable: Ietm, Add, Contains, Remove IDictionary: interface to implementStringsstring x = "abc"; or string x = @"abc"; # says treat string literally (ignore escape characters) string intStr = myInt.ToString(); Available methods: eg Copy, Compare, Format, SubString, ToUpper, Trim, Split (breaks into substrings) StringBuilder class for dynamically building and processing strings. Methods available: Append, Insert, Remove, Replace.Exception Handlingthrow new System.Exception (""); try { } catch { } or catch (Exception type) { } System.Exception provides Message(), StackTrace(), InnerException(),		IComparer: allows collection sorting
ArrayList: A dynamicall sized array. Use Add, Remove etc Queue: fifo collection. Use enqueue, dequeue, peek Stack: lifo collection. Use pop, push, peekDictionariesAssociates values with a key. Any kind of object can be associated with any type of key. Hashtable: Ietm, Add, Contains, Remove IDictionary: interface to implementStringsstring x = "abc"; or string x = @"abc"; # says treat string literally (ignore escape characters) string intStr = myInt.ToString(); Available methods: eg Copy, Compare, Format, SubString, ToUpper, Trim, Split (breaks into substrings) StringBuilder class for dynamically building and processing strings. Methods available: Append, Insert, Remove, Replace.Exception Handlingthrow new System.Exception (""); try { } catch { } or catch (Exception type) { } System.Exception provides Message(), StackTrace(), InnerException(),		
ArrayList: A dynamicall sized array. Use Add, Remove etc Queue: fifo collection. Use enqueue, dequeue, peek Stack: lifo collection. Use pop, push, peekDictionariesAssociates values with a key. Any kind of object can be associated with any type of key. Hashtable: Ietm, Add, Contains, Remove IDictionary: interface to implementStringsstring x = "abc"; or string x = @"abc"; # says treat string literally (ignore escape characters) string intStr = myInt.ToString(); Available methods: eg Copy, Compare, Format, SubString, ToUpper, Trim, Split (breaks into substrings) StringBuilder class for dynamically building and processing strings. Methods available: Append, Insert, Remove, Replace.Exception Handlingthrow new System.Exception (""); try { } catch { } or catch (Exception type) { } System.Exception provides Message(), StackTrace(), InnerException(),	Collection Types	
Stack: lifo collection. Use pop, push, peekDictionariesAssociates values with a key. Any kind of object can be associated with any type of key. Hashtable: Ietm, Add, Contains, Remove IDictionary: interface to implementStringsstring x = "abc"; or string x = @"abc"; # says treat string literally (ignore escape characters) string intStr = myInt.ToString(); Available methods: eg Copy, Compare, Format, SubString, ToUpper, Trim, Split (breaks into substrings) StringBuilder class for dynamically building and processing strings. Methods available: Append, Insert, Remove, Replace.Exception Handlingthrow new System.Exception (""); try { } catch { } or catch (Exception type) { } finally { } System.Exception provides Message(), StackTrace(), InnerException(),	<i>,</i> ,	
Dictionaries Associates values with a key. Any kind of object can be associated with any type of key. Hashtable: Ietm, Add, Contains, Remove IDictionary: interface to implement Strings string x = "abc"; or string x = @"abc"; # says treat string literally (ignore escape characters) string intStr = myInt.ToString(); Available methods: eg Copy, Compare, Format, SubString, ToUpper, Trim, Split (breaks into substrings) StringBuilder class for dynamically building and processing strings. Methods available: Append, Insert, Remove, Replace. Exception Handling throw new System.Exception(""); try { } catch { } or catch (Exception type) { } finally { } System.Exception provides Message(), StackTrace(), InnerException(),		
Exception Handlingtype of key. Hashtable: Ietm, Add, Contains, Remove IDictionary: interface to implementStringsstring x = "abc"; or string x = @"abc"; # says treat string literally (ignore escape characters) string intStr = myInt.ToString(); Available methods: eg Copy, Compare, Format, SubString, ToUpper, Trim, Split (breaks into substrings) StringBuilder class for dynamically building and processing strings. Methods available: Append, Insert, Remove, Replace.Exception Handlingthrow new System.Exception(""); try { } catch { } or catch (Exception type) { } finally { } System.Exception provides Message(), StackTrace(), InnerException(),		Stack: 1110 collection. Use pop, pusn, peek
Exception Handlingtype of key. Hashtable: Ietm, Add, Contains, Remove IDictionary: interface to implementStringsstring x = "abc"; or string x = @"abc"; # says treat string literally (ignore escape characters) string intStr = myInt.ToString(); Available methods: eg Copy, Compare, Format, SubString, ToUpper, Trim, Split (breaks into substrings) StringBuilder class for dynamically building and processing strings. Methods available: Append, Insert, Remove, Replace.Exception Handlingthrow new System.Exception(""); try { } catch { } or catch (Exception type) { } finally { } System.Exception provides Message(), StackTrace(), InnerException(),	Distinguis	Associated values with a key. Any kind of object can be associated with any
Hashtable: Ietm, Add, Contains, Remove IDictionary: interface to implementStringsstring x = "abc"; or string x = @"abc"; # says treat string literally (ignore escape characters) string intStr = myInt.ToString(); Available methods: eg Copy, Compare, Format, SubString, ToUpper, Trim, Split (breaks into substrings) StringBuilder class for dynamically building and processing strings. Methods available: Append, Insert, Remove, Replace.Exception Handlingthrow new System.Exception(""); try { } catch { } or catch (Exception type) { } finally { } System.Exception provides Message(), StackTrace(), InnerException(),	Dictionaries	
IDictionary: interface to implementStringsstring x = "abc"; or string x = @"abc"; # says treat string literally (ignore escape characters) string intStr = myInt.ToString(); Available methods: eg Copy, Compare, Format, SubString, ToUpper, Trim, Split (breaks into substrings) StringBuilder class for dynamically building and processing strings. Methods available: Append, Insert, Remove, Replace.Exception Handlingthrow new System.Exception(""); try { } catch { } or catch (Exception type) { } finally { } System.Exception provides Message(), StackTrace(), InnerException(),		
Strings string x = "abc"; or string x = @"abc"; # says treat string literally (ignore escape characters) string intStr = myInt.ToString(); Available methods: eg Copy, Compare, Format, SubString, ToUpper, Trim, Split (breaks into substrings) StringBuilder class for dynamically building and processing strings. Methods available: Append, Insert, Remove, Replace. Exception Handling throw new System.Exception(""); try { } catch { } or catch (Exception type) { } finally { } System.Exception provides Message(), StackTrace(), InnerException(),		
<pre>escape characters) string intStr = myInt.ToString(); Available methods: eg Copy, Compare, Format, SubString, ToUpper, Trim, Split (breaks into substrings) StringBuilder class for dynamically building and processing strings. Methods available: Append, Insert, Remove, Replace. Exception Handling throw new System.Exception(""); try { } catch { } or catch (Exception type) { } finally { } System.Exception provides Message(), StackTrace(), InnerException(),</pre>		
<pre>escape characters) string intStr = myInt.ToString(); Available methods: eg Copy, Compare, Format, SubString, ToUpper, Trim, Split (breaks into substrings) StringBuilder class for dynamically building and processing strings. Methods available: Append, Insert, Remove, Replace. Exception Handling throw new System.Exception(""); try { } catch { } or catch (Exception type) { } finally { } System.Exception provides Message(), StackTrace(), InnerException(),</pre>	Strings	string $x = $ "abc"; or string $x = @$ "abc"; # says treat string literally (ignore
Available methods: eg Copy, Compare, Format, SubString, ToUpper, Trim, Split (breaks into substrings) StringBuilder class for dynamically building and processing strings. Methods available: Append, Insert, Remove, Replace. Exception Handling throw new System.Exception(""); try { } catch { } or catch (Exception type) { } finally { } System.Exception provides Message(), StackTrace(), InnerException(),	Stilligs	escape characters)
<pre>(breaks into substrings) StringBuilder class for dynamically building and processing strings. Methods available: Append, Insert, Remove, Replace. Exception Handling throw new System.Exception(""); try { } catch { } or catch (Exception type) { } finally { } System.Exception provides Message(), StackTrace(), InnerException(),</pre>		<pre>string intStr = myInt.ToString();</pre>
StringBuilder class for dynamically building and processing strings. Methods available: Append, Insert, Remove, Replace. Exception Handling throw new System.Exception(``"); try { } catch { } or catch (Exception type) { } finally { } System.Exception provides Message(), StackTrace(), InnerException(),		
available: Append, Insert, Remove, Replace. Exception Handling throw new System.Exception(""); try { } catch { } or catch (Exception type) { } finally { } System.Exception provides Message(), StackTrace(), InnerException(),		
Exception Handling throw new System.Exception(""); try { } catch { } or catch (Exception type) { } finally { } System.Exception provides Message(), StackTrace(), InnerException(),		
<pre>try { } catch { } or catch (Exception type) { } finally { } System.Exception provides Message(), StackTrace(), InnerException(),</pre>		avallable: Appena, insert, kemove, Keplace.
<pre>try { } catch { } or catch (Exception type) { } finally { } System.Exception provides Message(), StackTrace(), InnerException(),</pre>	Evention Handling	throw new System Exception (" ").
<pre>catch { } or catch (Exception type) { } finally { } System.Exception provides Message(), StackTrace(), InnerException(),</pre>	Exception Handling	
<pre>finally { } System.Exception provides Message(), StackTrace(), InnerException(),</pre>		
System.Exception provides Message(), StackTrace(), InnerException(),		
TargeLSILe()		TargetSite()
Custom exceptions must derive from System.ApplicationException		
InnerException allows Exception to be saved as part of throwing a new		

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