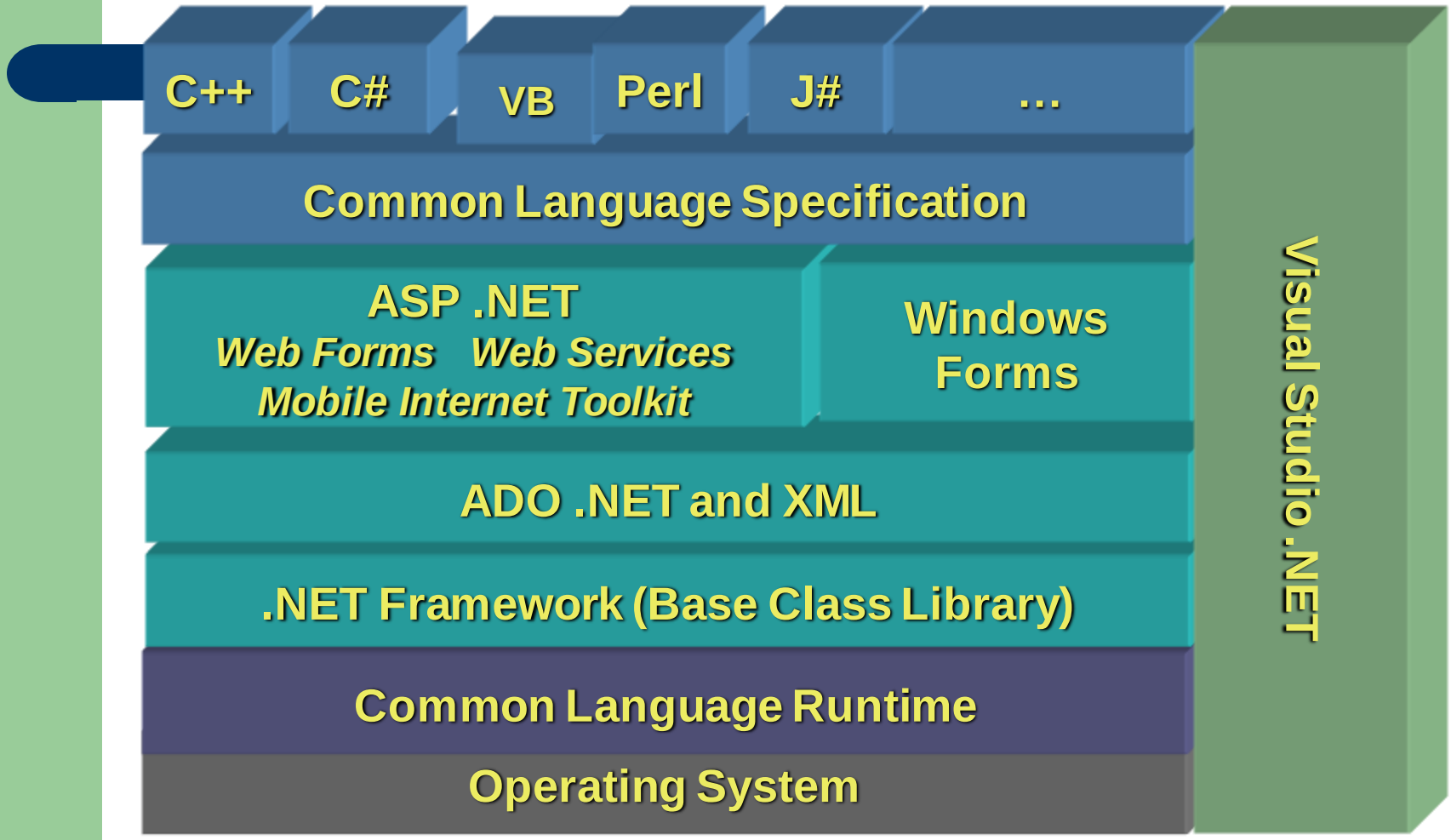
Programming in C#

So What is .NET

.NET is the Microsoft Web services strategy to connect information, people, systems, and devices through software

- .NET is a platform that provides a standardized set of services.
  - Data access and connectivity (ADO.NET)
  - User Interfaces (WinForms, WPF)
  - Web Applications (ASP.NET, Silverlight)
  - Network Communication (WCF), Workflow (WF)

.NET Framework



Core of the Framework : FCL & CLR

- Common Language Runtime
  - Garbage collection
  - Language integration
  - Multiple versioning support
    (no more DLL hell!)
  - Integrated security

Core of the Framework : FCL & CLR

- Framework Class Library
  - Provides the core functionality:

    ASP.NET, Web Services, ADO.NET, Windows Forms, IO, XML, etc.

Versions of C#

- C# 1.0 - First Version
- C# 2.0 – Generics, Anonymous methods, Nullable types
- C# 3.0 – Implicit Typing, Object and Collection initializers, Lamba expressions
- C# 4.0 – Dynamic Typing, Optional parameters, Named Arguments
- C# 5.0 – Asynchronous functions

.NET and Visual Studio Versions

- 2002 - .Net 1.0 / Visual Studio.NET
- 2003 - .Net 1.1 / Visual Studio 2003
- 2005 - .Net 2.0 / Visual Studio 2005
- 2007 - .Net 3.5 / Visual Studio 2008
- 2008 - .Net 3.5sp1

.NET and Visual Studio Versions

- 2010 - .Net 4.0/ VS.Net 2010
- 2012 - .Net 4.5/VS.Net 2012 , VS.Net 2013(support for Windows 8.1 Apps development)
- 2015 - .Net 4.6/VS.Net 2015

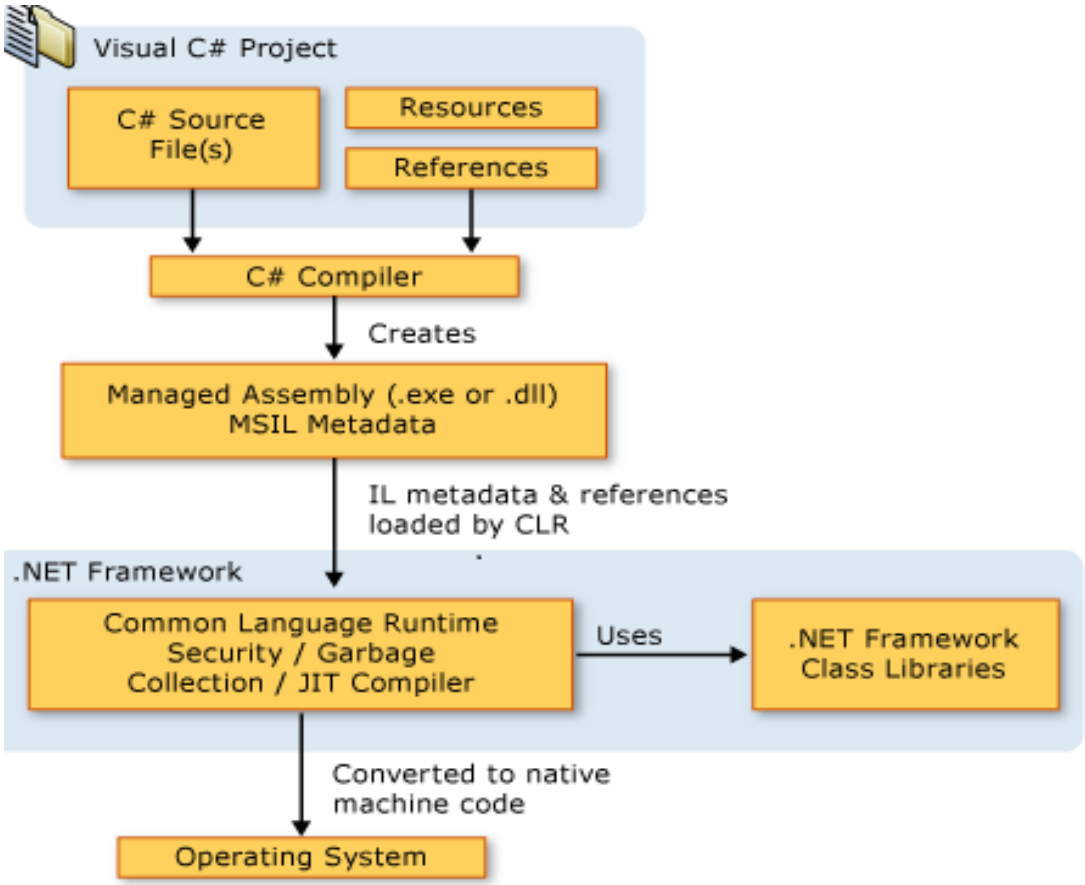
CLS – Common Language Specification

- Provides basic set of features to be implemented
- CLS enables language interoperability
- Components that adhere to CLS rules are called CLS-compliant
- CTS is a subset of CLS

CTS – Common Type System

- It defines the rules which Common Language Runtime follows when declaring, using, and managing types
  - Enables cross language integration
  - Provides an object oriented model for implementation by many languages
  - Defines rules that every language must follow under .NET framework

C# Code Compilation



C# Program Structure

- Namespace declaration
- A Class
- Class methods
- Class attributes
- A Main method
- Statements & Expressions
- Comments

Namespace

- Define Namespace

```
namespace namespace_name
{
        // code declarations
}
```

- Reference item inside namespace

```
namespace_name.item_name;
```

- The *using* keyword
- Nested namespace

Access Specifiers

- Encapsulation – enclosing in physical or logical package
- Encapsulation is implemented by access specifiers
- The access specifiers are
  - Public, Private, Protected, Internal and Protected Internal

Access Specifiers

- Public
  - Any public member can be accessed from outside the class
- Private
  - Only functions of the same class can access private members

Access Specifiers

- Protected
  - Allows inherited class to access the member variables and functions of base class
- Internal
  - Is made available to other functions and objects in the current assembly.
- Protected Internal
  - Only available to inheriting classes of the application

Assemblies

- Fundamental unit of deployment, version control , reuse and security for a .net application
- Logical unit that aids in distribution
- Takes the form of .dll or .exe
- Assembly contains data about elements in the assembly
- Can contain single or multiple files.

Assemblies

- Assembly manifest contains the assembly metadata
- Assemblies can be loaded side-by-side
- Assembly contains a PE Header , CLR Header Metadata and IL code and Resources.

Assemblies

- IL Disassembler can be used to view the IL code

  Ildasm <AssemblyName>
- Assemblies can be private or shared
- Shared assemblies are placed in the GAC

Global Assembly Cache

- Machine-wide central repository of assemblies
- Assemblies in GAC must be strongly named
- Strong name consist of its simple text name, version number, and culture information (if provided)—plus a public key and a digital signature

Global Assembly Cache

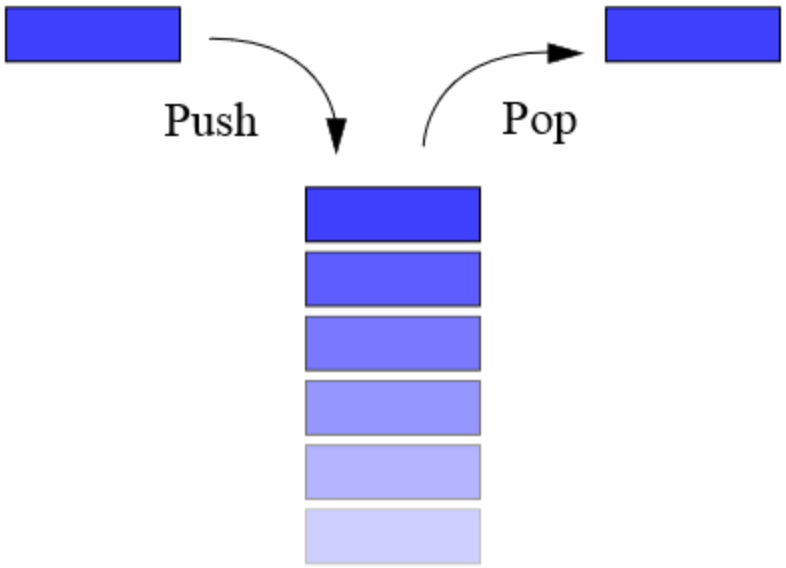
- Global Assembly Cached Tool
  - gacutil.exe
- Location of GAC
  - %windir%\Microsoft.NET\assembly\

Data Types

- Value Types
  - Derive from System.ValueType
- Reference Types
  - Contains reference to a variable
- Pointer Types
  - Pointer type variables store the memory address of another type

Stack Memory

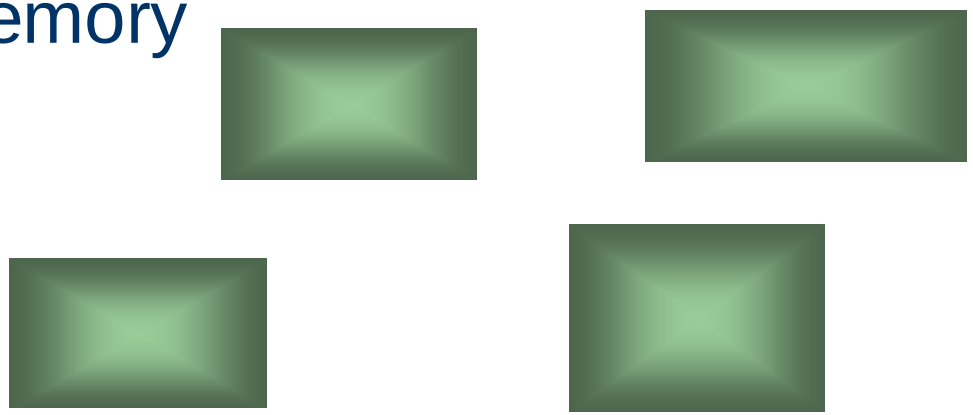
- Keeps track of code execution
- Stack takes care of its own memory management



Heap Memory

- Heap is used to store objects
- Allows access and is not restricted to any order
- Reference Type always goes into heap

Heap Memory



Value Type

- Variables of Value Type contain the data
- Built-in data types
- Stored in the stack
- Memory reclaimed after usage

Reference Type

- Variables of reference type store reference to their data
- The following keywords are used to declare reference types:

  class, interface ,delegate
- C# also provides the following built-in reference types:

  dynamic, object ,string

Reference Type

- Reference Type is stored in the Heap
- Memory is reclaimed by Garbage Collection.

Structures

- structs are value types
- structs get created on the stack
- structs can be called without using new operator
- There is no inheritance for structs as there is for classes

Boxing

- Value type is converted to reference type

```
int i = 67; // i is a value type
object o = i; // i is boxed
System.Console.WriteLine(i.ToString()); // i is boxed
```

Unboxing

- Unboxing is seen in classes designed to use for objects

```
System.Collections.ArrayList list =
                    new System.Collections.ArrayList();
  int n = 67; // n is a value type
  list.Add(n); // n is boxed
  n = (int)list[0]; // list[0] is unboxed
```

Operators

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Misc Operators

Miscellaneous Operators

sizeof()	Returns size of a data type
typeof()	Returns type of a class
&	Returns the address of a variable
*	Pointer to a variable
?:	Conditional Expression
is	Determines if object of certain type
as	Cast without exception if cast fails

Conversion and Cast

- Conversion occurs when it is type safe and there is no data loss
- Explicit conversion is called cast
- Run time errors can occur during explicit cast

Decision Making

- If statement
- If else statement
- Switch statement

Decision Making

- Switch Statement

```
switch(expression)
{ case constant-expression :
        statement(s);
        break;
case constant-expression : statement(s);
      break;
     /* you can have any number  of case statements */
default :
statement(s);
}
```

Loops

- while loop
- do while loop
- for loop
- Nested loops
- Loop Control
  - break
  - continue

Strings

- String and string are same and they can be used interchangeably
- Strings are immutable
- String manipulation tips
  - Use StringBuilder to concatenate a large number of strings
  - Avoid using == and != in String comparison instead use String.Equals

Arrays

- C# Arrays are zero indexed
- There are three types
  - Single Dimensional arrays
  - Multi Dimensional arrays
  - Jagged arrays (Array of arrays)

Arrays

- Arrays are objects in C#
- System.Array is the base type for all arrays
- Array is to store multiple variables of the same type

Implicitly Typed Variable

- var i = 10; // implicitly typed
- int i = 10; //explicitly typed
- Local variables can be inferred as var
- The type is inferred from the expression on the right side

Anonymous Types

- Anonymous types encapsulates the set of read only properties into a single object without explicitly defining a type first.
- Cannot declare field, property, an event or the return type of a method as Anonymous Type.

Implicitly Typed Arrays

- The type of the array instance is referred from the elements specified in the array initializer

```
var a = new[] { 1, 10, 100, 1000 };
```

Reflection

- Reflection is the ability of a managed code to read its own metadata
- Use GetType() method to get the type of the current instance.
- Use GetMethod() to check the existence of a method
- Include the namespace System.Reflection

Class

- Class is a blueprint for a data type
- Objects are instances of the class
- Methods and variables are called the members of the class
- Class has a default constructor and destructor

Class

- The default access specifier for a class is internal
- The default access specifier for the members is private

Class Syntax

```
<access specifier> class  class_name {
    // member  variables
    <access specifier> <data type> variable1;
    <access specifier> <data type> variable2;
    ...
 // member  methods
   <access specifier> <return type> method1(parameter_list)
    {
        // method body
    }
   …
}
```

Properties

- Property is a member variable that provides access to private members of the field
- Properties are called accessors

Constructor

- Constructor is executed whenever a new object is created
- Constructor has the same name as class and does not have a return type.
- Default constructor has no parameters
- We can parameterize the constructor if needed.

Destructor

- Destructor of a class is executed when the object goes out of scope
- Destructor will have the exact name as the class prefixed with a tilde (~)
- It can neither take in parameters nor can it return value.

Static Members

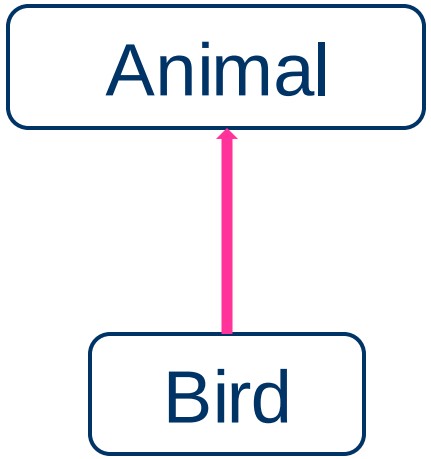
- Static means only one instance of the variable exists
- Static Variables are used for declaring constants
- They can be retrieved without instantiating the class

Inheritance

- Inheritance helps create new classes that reuse , extend and modify the base class
- The class whose members are inherited are called the base class
- The class that inherits is called the derived class
- Inheritance is transitive.

Inheritance

- Inheritance should create an ***is-a*** relationship, meaning the child *is a* more specific version of the parent and is depicted as follows



Inheritance

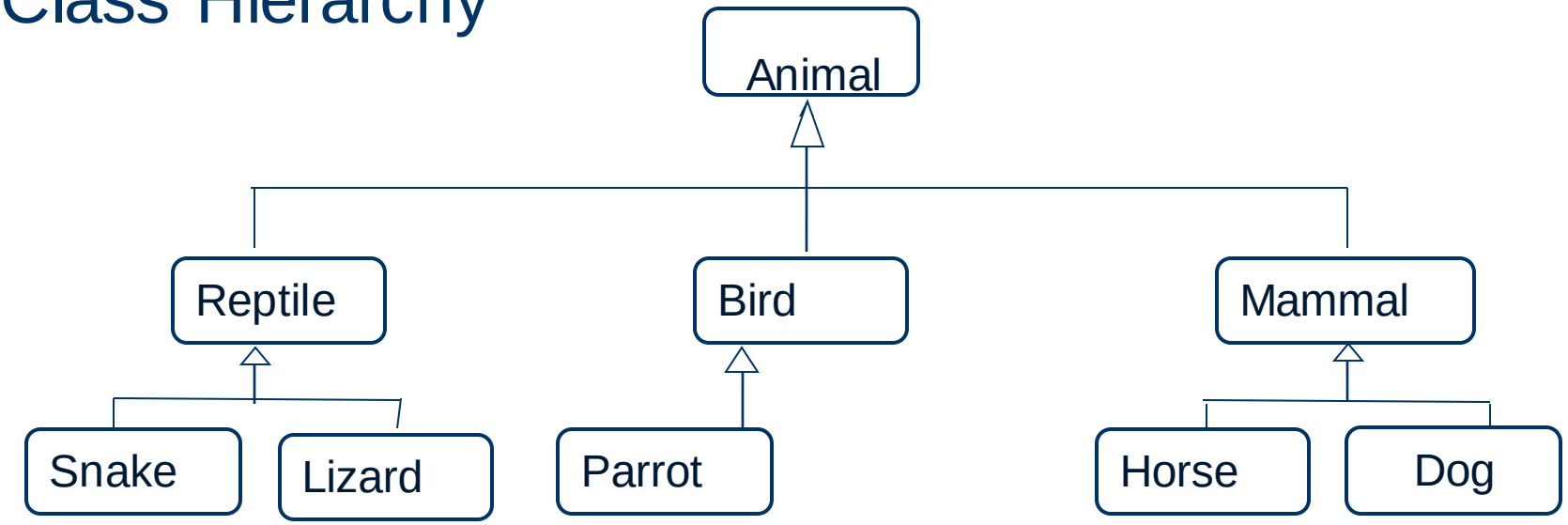
Base Class	Derived Class
Student	Graduate Student Undergraduate Student
Shape	Circle Rectangle Triangle
Loan	CarLoan HomeLoan
Account	CheckingAccount SavingsAccount

Inheritance

- C# supports single inheritance
- Private variables are not accessible in the child class
- Public and protected variables are accessible in the child class

Inheritance

Class Hierarchy



Reference and Inheritance

- An object reference can refer to an object of its class, or to an object of any class derived from it by inheritance.
- For example, if the `Holiday` class is used to derive a child class called `Christmas`, then a `Holiday` reference can be used to point to a `Christmas` object.

```
Holiday day = new Holiday();
Day = new Christmas();
```

Interfaces

- Group of related functions that a class or struct can implement
- Interfaces can be used to mimic the behaviour of multiple inheritance
- Interfaces can contain properties, events, methods or indexers

Interfaces

- Explicitly implemented members can be accessed through the interface instance
- Indexers can be declared on interfaces

Abstract Classes

- Representative classes out of which you would not like to and cannot create objects.
- Abstract Classes can contain abstract methods and non abstract methods
- Abstract Classes can contain abstract properties

Abstract Classes Rules

- Abstract Class cannot be sealed
- Abstract Methods
  - cannot be private
  - cannot be virtual
  - cannot be static

Sealed Class

- Sealed Modifiers prevents other classes from inheriting it.
- Abstract Modifier cannot be used with sealed modifier

Polymorphism

- Polymorphism means many shaped or many forms
- Polymorphism can be static or dynamic
- Static polymorphism response to a function is determined at compile time
- Dynamic Polymorphism the response to a function is decided at run-time

Polymorphism

- Static polymorphism can be implemented using two techniques
  - Function Overloading
  - Operator Overloading
- Function overloading allows multiple functions with the same name provided they have a difference in parameters and or return type

Polymorphism

- Dynamic Polymorphism can be achieved through virtual and override methods
- Base class members can be hidden using the new keyword
- Preventing derived classes from overriding a method using sealed keyword
- Accessing base class members using the base keyword

IEnumerable and IEnumerable<T>

- IEnumerable supports iteration of a non generic collection
- Exposes the GetEnumerator method
- IEnumerable<T> supports iteration of a generic collection
- Exposes the GetEnumerator<T> method

IEnumerator and IEnumerator<T>

- IEnumerator supports simple iteration over a non-generic collection
- It has a property Current
- Exposes methods MoveNext and Reset
- foreach is used to iterate an Array or Collection that implements IEnumerable interface and hides complexity
- IENumerator<T> is the generic version.

yield Keyword

- yield is used to indicate that the method in which it appears is an iterator
- yield preserves the state of iteration
- yield return is used to return each element one at a time
- yield break on the other hand is used to end the iteration

Dynamic

- It bypasses compile time type checking
- With dynamic, calls can be made to non existent methods
- The check happens at run time
- It is useful when working with external libraries.

Async and Await

- Asynchrony is essential for methods that are blocking
- Asynchronous non blocking methods can be accomplished with async and await
- The application can continue with other work that does not depend on the web resource when async method executes

Async and Await

Application area	Supporting APIs that contain async methods
Web access	HttpClient , SyndicationClient
Working with files	StorageFile, StreamWriter, StreamReader, XmlReader
Working with images	MediaCapture, BitmapEncoder, BitmapDecoder
WCF programming	Synchronous and Asynchronous Operations

Async and Await

- The method signature includes async keyword
- The name of the asyc method by convention ends with “Async”
- An async method returns void or Task or Task<TResult>

Async and Await

- The Main method cannot be async and it cannot use the await keyword.It must start an async method with await
- Async method without await is synchronous

Named and Optional Arguments

- Associating argument with parameter name instead of the parameter position
- Optional arguments enables omitting arguments for some parameters
- Both can be used with methods, indexers, constructors and delegates.

Generics

- What are Generics?
  - Generics is a way to let you to define type-safe classes without compromising type safety, performance, or productivity
  - use the **<** and **>** brackets, enclosing a generic type parameter

Generics

- Generics allow deferring specification of types until it is actually used
- Most common use is to create collection classes
- Generics maximize code reuse, type safety and performance.
- Can create generic interface, classes, methods, events and delegates

Generics

- Multiple Generic Types are allowed in classes
- Aliasing is permitted in generics
- Implement the ICompare Interface to search the Generic Type.

Generics

- Generics lets you reuse code
- Improves performance as it does not enforce boxing and unboxing
- Generic Delegates enable type safe call back

Generics

- You can constrain the Generic to be a value type or reference type

```
public class MyClass<T> where T : struct {...}
public class MyClass<T> where T : class {...}
```

- You can implicitly cast Generics to Object
- Explicit casting can also be forced

Generics

- Provide type argument when deriving from generics base class

```
public class BaseClass<T> {...}
public class SubClass : BaseClass<int> {...}
```

- If SubClass is also generic

```
public class SubClass<T> : BaseClass<T> {...}
```

Generics

```
public class Calculator<T>
{
public T Add(T arg1,T arg2)
 { return arg1 + arg2;//Does not compile }
}
```

- Above Code does not compile
- Have an abstract base class with Generic and then implement the methods in the concrete inheriting class

Delegates

- A delegate is a reference type variable that holds reference to a method
- Similar to function pointers
- Syntax

  delegate <return type> <delegate-name> <parameter list>

- Example

  public delegate int MyDelegate (string s);

Delegates

- Delegates can be associated with any method with a compatible signature and return type
- Declared delegates must be instantiated with a new keyword

```
public delegate void printString(string s);
...
printString ps1 = new printString(WriteToScreen);
printString ps2 = new printString(WriteToFile);
```

Delegates

- Delegate objects can be composed using “+” operator
- A composed delegate calls the two delegates it was composed from
- “-” is used to remove a component delegate from the composed delegate
- Only delegates of the same type can be composed

Anonymous Methods

- Anonymous methods provide a way to provide a code block as a delegate parameter
- They are methods without name just the body
- Example

```
delegate void NumberChanger(int n);
...
NumberChanger nc = delegate(int x) {
Console.WriteLine("Anonymous Method: {0}", x); };
```

Events

- Events enable a class or object to notify other classes when something occurs
- The class that raises the event is publisher
- The class that handles the event is subscriber

Events

- Publisher determines when an event is raised
- Events can have multiple subscribers
- Subscribers can handle events from multiple publishers
- Events that have no subscribers are never raised

Exception Handling

- Exception is a problem that occurs during program execution
- It provides a way to transfer control from one part of a program to another
- C# exception handling is based on four keywords namely
  - try, catch, finally and throw

Exception Handling

```
try {
// statements causing exception }
 catch( ExceptionName e1 )
 { // error handling code }
 catch( ExceptionName e2 )
 { // error handling code }
 catch( ExceptionName eN )
{ // error handling code }
 finally { // statements to be executed }
```

Exception Handling

Exception Class	Description
System.IO.IOException	Handles I/O errors.
System.IndexOutOfRangeException	Handles errors generated when a method refers to an array index out of range.
System.ArrayTypeMismatchException	Handles errors generated when type is mismatched with the array type.
System.NullReferenceException	Handles errors generated from deferencing a null object.
System.DivideByZeroException	Handles errors generated from dividing a dividend with zero.
System.InvalidCastException	Handles errors generated during typecasting.
System.OutOfMemoryException	Handles errors generated from insufficient free memory.
System.StackOverflowException	Handles errors generated from stack overflow.

Exception Handling

- Trace of all the methods in the execution stack is called stack trace.
  - Exception.StackTrace
- Difference between throw and throw ex
  - throw ex resets the stack trace
  - throw preserves the stack trace

XML

- XML is a popular choice for a wide variety of applications due to its interoperability
- Document structure and content validation can be done using XML schema
- XML Style Sheet for transformation (XSLT) enables document transformation

XML

- Application reads information from XML document is known as parsing
- Microsoft is a big proponent of XML
  - Query results can be returned as XML
  - Configuration files are in XML
  - Web Services use XML based SOAP

XML and .NET

- System.Xml namespace
  - System.Xml contains essential classes for reading and writing
- This in turn contains
  - System.Xml.Schema
  - System.Xml.XmlPath
  - System.Xml.Serialization
  - System.Xml.Xsl
  - System.Xml.Linq

XML and .NET

- There are three different techniques of parsing
  - XML readers and writers
  - XML document editing using DOM
  - XML document editing using XPathNavigator

File I/O

- When a file is opened for reading or writing it becomes a stream
- Stream is a sequence of bytes passing thru a communication path
- There is an input stream and an output stream

I/O Classes

BinaryReader	**Reads primitive data from a binary stream.**
BinaryWriter	**Writes primitive data in binary format.**
BufferedStream	**A temporary storage for a stream of bytes.**
Directory	**Helps in manipulating a directory structure.**
DirectoryInfo	**Used for performing operations on directories.**
DriveInfo	**Provides information for the drives.**
File	**Helps in manipulating files.**

I/O Classes

- FileInfo Used for performing operations on files.
- FileStream Used to read from and write to any location in a file.
- MemoryStream Used for random access to streamed data stored in memory.
- Path Performs operations on path information.
- StreamReader Used for reading characters from a byte stream.
- StreamWriter Is used for writing characters to a stream.
- StringReader Is used for reading from a string buffer.
- StringWriter Is used for writing into a string buffer.

ADO.NET

- Provides a consistent way to access datasources
- Datasources could be exposed through OLEDB or ODBC
- The ADO.NET classes can be found in the System.Data namespace
- For higher level of abstraction ADO.NET Entity Framework

ADO.NET

- ADO.NET Objects provided are
  - Connection Object
  - Command Object
  - DataReader
  - DataAdapter
  - DataSet

ADO.NET

- DataSet is a disconnected database
- It contains one or more Datatables
- The DataTable can have row, column, primary and foreign key constraint

ADO.NET

- DataReader Vs DataSet

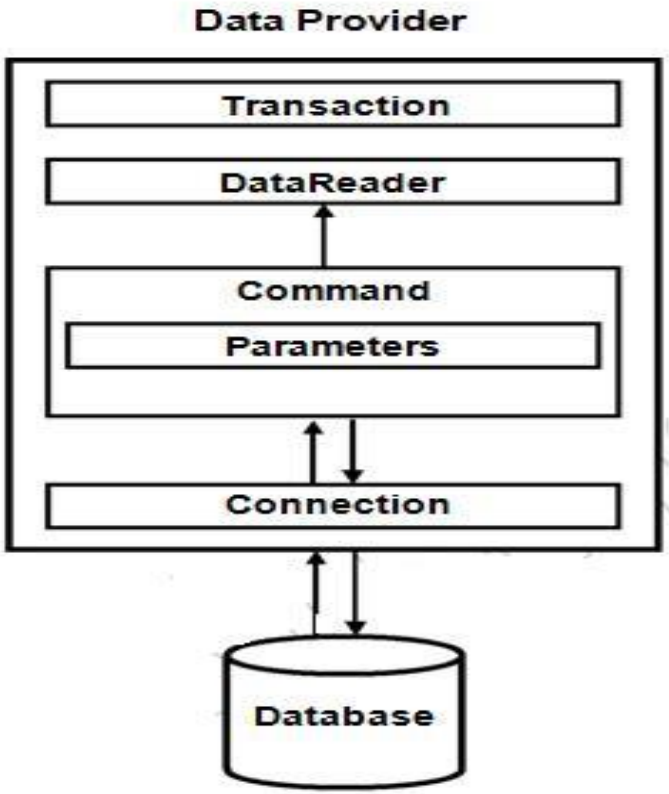
  DataSet is used

  - To cache data locally in your application
  - Pass Data between tiers of the application
  - Extensive processing on Data
  - Dynamically interact with data like binding it to a control

ADO.NET

- DataReader is used to only read results of the query
- By using only DataReader performance can be boosted.

ADO.NET – Connected Architecture



ADO.NET – Disconnected Architecture