Contents

[C# Programming Language 1](#_Toc63029333)

[WPF 2](#_Toc63029334)

[ASP.NET 3](#_Toc63029335)

C# Programming Language – An object-oriented programming language.

* It is a strongly typed language. Every variable, constant, expressions that return a value have a type. And after you declare a variable it can’t be declared with a new type. The compiler checks for type safety.

Managed code is code that runs under the control of CLR.

.NET – a virtual execution system.

CLI – Common Language Infrastructure

CLR – Common Language Runtime

CTS – All types, including numeric types, derive ultimately from a single base type “System.Object”. This unified type hierarchy is called the Common Type System (CTS).

CIL –

**TYPES AND MEMBERS**

Classes and Objects – A class is a data structure that combines state (fields) and actions (methods) in a single unit. Instances of a class are objects.

Type parameters – Generic classes define type parameters.[ e.g. List<TYPE\_PARAM> ] Type parameters are a list of type parameter names enclosed in angle brackets. They follow the class name and are used in the body of the class to define members of the class.

Structs – Struct types are simpler data structures designed to store data values. They don’t support inheritance.

Interface – defines a contract that can be implemented by classes and structs. It can contain methods, properties, events and indexers.

Enums – Enums define a set of constant values. They can be used in combination as flags.

Nullable types – nullable variables can hold an additional null value, indicating no value. They are represented by System.Nullable<T>. or int? nullableInt = null;

Tuples – provides a concise syntax to group multiple data element in a lightweight data structure. E.g

(double Sum, int Count) t2 = (4.5, 3);

Indexer – is a member that enables objects to be indexed in the same way as an array. It is declared using “this[]”.

Expression – are constructed from operands and operators (these are **+**, **-**, **\***, **/**, **new** etc).

Statements – Actions of a program are expressed using statements.

* **BLOCK** – A list of statements written between the delimiters “{“ “}”.
* **DECLARATIONS STATEMENTS** – are used to declare local variables and constants.
* **EXPRESSION STATEMENTS** – Are used to evaluate expressions.
* **SELECTION STATEMENTS** – for example the “if” and “switch” statements.
* **ITERATION STATEMENTS** – while, do, for and foreach statements.
* **JUMP STATEMENTS** – Used to transfer control. Eg break, continue, goto, throw, return and yield.
* **TRY CATCH STATEMENTS -**
* **CHECK AND UNCHECKED STATEMENTS**
* **LOCK STATEMENT –** Used to obtain the mutual exclusion lock of a given object, execute a statement and then release the lock.
* **USING STATEMENT –** Used to obtain a resource, execute a statement, and then dispose of that resource.

**NULLABLE REFERENCE TYPES**

C# CONCEPTS

**C# TYPE SYSTEM**

**Topics**

* **Built-in types**
* **Custom types**
* **Value types**
* **Reference Type**
* **Types of literal values**
* **Generic types**
* **Implicit types**
* **Anonymous Types**
* **Nullable value types**
* **Compile-time type and Runtime type**

**BUILT-IN TYPES**

**Value Types**

bool System.Boolean

byte System.Byte

sbyte System.S

char

decimal

double

float

int

uint

long System.Long

ulong

short

ushort

**Reference Types**

object

string

dynamic

**CUSTOM TYPES**

Use struct, class, interface and enum to create your own custom types.

**NULLABLE REFERENCE TYPES**

**Topics**

* **Nullability of types**
* **Nullable contexts**
* **Nullable annotation context**

Starting from C# 8.0.

A nullable reference type is noted using the same syntax as nullable value types. E.g. string? name;

Nullability of types – a reference type can have one of four nullabilities.

* **Nonnullable**: null can’t be assigned.
* **Nullable**: null can be assigned to these variables.
* **Oblivious**: the pre C# 8.0 state.
* **Unknown**: for type-parameters where constraints don’t specify that the type must be nullable or nonnullable.

**NAMESPACES**

NAMESPACES – Used to organize classes. Using keyword is used so that the complete name is not required.

**BASIC TYPES**

Types are declared using their **type** or **var** keyword.

**Built-in Types** – C# provides built in types for numbers, Booleans and text.

**Custom Types** – Use struct, class, interface and enum to create your own custom types.

**Generic Types -**  A type with one or more type parameters.

**Implicit types**

**Anonymous types**

**Tuple type**

**CLASSES**

* Declaring classes
* Creating Objects
* Class Inheritance

**DECONSTRUCTING TUPLES**

* Defining a tuple
* Deconstructing a tuple
* Deconstructing with discards
* Deconstructing user-defined types
* Deconstructing a user-defined type with discards.
* Deconstructing a user-defined type with and extension method.

**INTERFACES**

An interface contains definitions for a group of related functionalities that a non-abstract class or a struct must implement.

**METHODS**

A method is a block of code that contains a series of statement.

**PROPERTIES**

Properties are first class citizens. They behave like fields when they are accessed, but are implemented with accessors that define the statements executed when a property is accessed or assigned.

\*\*

**INDEXERS**

Indexers are similar to properties. They enable indexed properties: properties referenced using one or more arguments. Those arguments provide an index into some collection of values.

public int this[string key]

{

get {return storage.Find(key); }

set {storage.SetAt(key, value); }

}

var item = someObject["key"];

someObject["AnotherKey"] = item;

**DISCARDS**

Placeholder variables that are intentionally unused in application code.

**GENERICS**

Generics introduce the concept of type parameters to .NET which make it possible to design classes and methods that defer the specification of one or more types until the class or method is declared or instantiated by client code.

**ITERATORS**

Iterating with foreach.

C# enables you to build methods that create a source for an enumeration, or iterator methods. An iterator method defines how to generate the objects in a sequence when requested.

You use yield return contextual keywords to define an iterator method.

**DELEGATES AND EVENTS**

Delegates provide a late binding mechanism in .NET. (Late binding means that you create an algorithm where the caller also supplies at least one method that implements part of the algorithm).

public delegate int Compare<in T>(T Left, T right);

implements

Events are also a late binding mechanism.

Events are a way for an object to broadcast that something has happened. Any other component can subscribe to that event and be notified when the event is raised.

public event EventHandler <FileListArgs>Progress;

**LANGUAGE INTEGRATED QUERY LINQ**

A technology that integrates query capabilities directly into the C# language.

**ASYNCHRONOUS PROGRAMMING**

C# has a language level asynchronous programming model, which allows for easily writing asynchronous code without having to juggle callbacks or using a library.

**PATTERN MATCHING**

Patterns test that a value has a certain shape, and can extract information from the value when it has the matching shape.

**WRITE SAFE, EFFICIENT CODE**

**EXPRESSION TREES**

Expression trees represent code as a structure that you can examine, modify or execute. These tools give you the power to manipulate code during run time.

**NATIVE INTEROPERABILITY**

Interoperability enables you to preserve and take advantage of existing investments in unmanaged code. .NET enables interoperability with unmanaged code through platform invoke services.

**VERSIONING**

THE .NET COMPILER PLATFORM SDK

(SKIPPED) – Will come back when I have time

C# PROGRAMMING GUIDE

**PROGRAMMING CONCEPTS**

**Asynchronous Programming –** the task asynchronous programming model, provides an abstraction over asynchronous code.

* **Attributes**

Attributes provide a powerful method of associating metadata, or declarative information with code (assemblies, types, method, properties and so forth). After an attribute has been associated with a program entity, the attribute can be queried at run-time using a technique called reflection.

[Serializable]

public class SampleClass

* **Collections**

Arrays are most useful for creating and working with a fixed number of strongly typed objects.

Collections however provide a more flexible way to work with groups of objects. The group of objects you work with can grow and shrink dynamically.

* **Convariance and Contravariance**

Covariance and contravariance enable implicit reference conversion for array types, delegate types, and generic type arguments. Covariance preservers assignment compatibility and contravariance reverses it.

* **Expression Trees**

Expression trees represent code in a tree-like structure where each node is an expression.

* **Iterators**

An iterator can be used to step through collections such as lists and arrays.

* **LINQ**

A technology that integrates query capabilities directly into the C# language.

* **Reflection**

Reflection provides objects (of type Type) that describe assemblies modules and types. You can use reflection to dynamically create an instance of a type, bind the type to an existing object, or get the type from an existing object. If you are using attributes in your code, reflection enables you to access them.

* **Serialization**

Serialization is the process of converting an object into a stream of bytes to store the object or transmit it to memory, a database, or a file.

**STATEMENTS, EXPRESSION AND OPERATORS**

C# code consists of statements made of keywords, expressions and operators.

**Statements**

The actions that a program takes are expressed in statements.

The order in which statements are executed in a program is called the flow of control or flow of execution.

A statement block is enclosed in {} and can contain nested blocks.

Types of statements

* Declarative statements – introduces a new variable.
* Expression statements – return a value.
* Selection statements – enable you to branch to different selections of code (if, else, switch, case)
* Iteration statements – enable you to loop through collections like arrays (do, for, foreach, in, while)
* Jump statements – transfer control to another section of code. (break, continue, default, goto, return, yield)
* Exception handling statements- enable you to gracefully recover from exception conditions that occur at run-time.
* Checked and unchecked – enable you to specify whether numerical operations are allowed to cause an overflow.
* Await statement – enables asynchronous method calls.
* The yield return statement – An iterator performs a custom iteration over a collection. The iterator uses the yield return statement to return each statement one at a time.
* The fixed statement – prevents the garbage collector from relocating a moveable variable.
* The lock statement – enables you to limit access to blocks of code to only one thread at a time.
* Labelled statement – you can give a statement and the use the goto keyword to jump to the labelled statement.
* The empty statement – consists of a single semicolon.

**Expression body definitions**

Expression body definitions let you provide a member’s implementation in a very concise, readable form. Used in a method or property definition is a single line. Eg.

method(param1, param2) => expression;

**Anonymous Functions**

An anonymous function is an “inline” statement or expression that can be used whenever a delegate type is expected.

They are created using lambda expressions or anonymous methods.

**Equality comparisons**

Value equality – Testing to see if two values compare.

Reference equality or identity – determines whether two variables refer to the same underlying object in memory.

Any struct that you define already has an implementation of value equality that it inherits from System.ValueType. It is recommended we override this feature.

To support reference equality comparisons, all types have a static Object.ReferenceEquals method.

**TYPES**

C# is a strongly typed language.

Use struct, class, interface and enum to create custom types.

* All types derive ultimately from a single base class type, System.Object. This unified type hierarchy is called the Common Type System (CTS).
* Each type in the CTS is defined as either a *value type* or a reference type.

Value types – Derive from System.ValueType. It’s variables directly contain their values, which means the memory is allocated inline in whatever context the variable is declared. There is no separate heap allocation or garbage collection overhead.

* **Structs** and **enums** are values types.
* Value types are sealed.
* A struct can implement interfaces.

Enum – an enum defines a set of named integral constants.

Reference Types – include types defined as a class, delegate, array or interface.

* When declared, it contains null, until an object is created using the new operator.
* When an object is created, Memory is allocated on the managed heap. The variable holds only a reference to the location of the object.
* An interface must be initialized with a class object that implements it.
* All Arrays are reference types, even if their elements are value types.
* They fully support inheritance.

Literal Values receive a type from the compiler. You can specify the type of a numeric literal by appending a letter to the end of the number.

Compile-time type and Runtime type. A variable can have different compile-time and run-time types. Compile-time type is the declared or inferred type of the variable in the source code. Run-time type is the type of the instance referred to by that variable.

Generic Types – A type declared with one or more type parameters that serve as placeholder(s) for the concrete type that client code will provide when it creates an instance of the type.

Implicit types – are declared using the var keyword and their type is provided by the compiler at compile time.

Anonymous types – Provide a convenient way to encapsulate a set of read-only properties into a single object without having to explicitly define a type first.

var v = new { Amount = 100, Message = “Hello” };

**CLASSES AND STRUCTS**

Classes and structs are data structures that encapsulate a set of data and behaviors (members) that belong together as a logical unit.

They are like blueprints that are used to create instances or objects at runtime.

A class is a reference type.

A struct is a value type. Structs are suited for small data structures that contain primarily data that is not intended to be modified after the struct is created.

Members Include – Fields, Constants, Properties, Methods, Constructors, Events, Finalizers, Indexers, Operators, Nested Types

**Classes** – a class is a reference type. It contains null when declared. Objects are created using the new keyword. An Object is a concrete entity based on a class.

Classes support single inheritance and implement interfaces. When a class declares a base class, it inherits all the members of the base class except the constructors.

Abstract classes declare abstract methods that have a signature definition but no implementation. They cannot be instantiated.

**Objects –** An object a block of memory that has been allocated and configured according to its blueprint class or structure. They are stored in a named variable or a collection. Class objects are created using a new keyword. Struct objects may be created by new but its not required.

**Object Identity vs Value Equality**

To determine whether two class instances refer to the same location in memory (which means they have the same identity) use the static Equals method.

To determine whether the instance fields in two struct instances have the same values, use the ValueTypes.Equals method.

**Inheritance –** Inheritance enables you to create new classes that reuse, extend, and modify the behavior defined in other classes. A derived class can have one direct base class.

The derived class implicitly gains all the members of the base class except for its constructors and finalizers.

A sealed class cannot be inherited from.

**Polymorphism –** takes two aspects:

* At run-time, objects of a derived class may be treated as objects of a base class in places such as method parameters, collections or arrays.
* Base classes may define and implement virtual methods which derived classes can override.

*Hide base class members with new members* – ie if you want your derived class to have a member with the same name as a member in your base class, you can use the new keyword to hide the base class member.

A derived class can stop a method from being overridden by declaring it as sealed.

**Members** – Fields, Constants, Properties, Methods, Constructors, Events, Finalizers, Indexers, Operators, Nested Types

Abstract Classes cannot the instantiated. They just provide a common definition of a base class.

Abstract methods have no implementation.

Static classes cannot be instantiated. You access the member of a static class by using the class name itself. A static constructor is only called once. It contains only static members. Is sealed.

Static members. A static member is callable on a class level. Only one copy of a static member exists. Static methods and properties cannot access non-static fields and events in their containing type. Static methods can be overloaded but not overridden. ( …………. )

**Access Modifiers –** Type and members

Public – Can be accessed by any other code in the assembly

Private – can be accessed only by code in the same class or struct.

Protected – can be accessed only by code in the same class, or in a class that is derived from that class.

Internal – can be accessed by any code in the assembly in which it’s declared, or from within a derived class in another assembly.

Private protected – can be accessed only within its declaring assembly, by code in the same class, or in a type that is derived from that class.

Classes and structs, declared within a namespace (ie **are not** nested within another type) can either be public or internal.

Derived classes **cannot** have greater accessibility than their base types.

You can enable specific other assemblies to access your internal types by using the **InternalsVisibleToAttribute.**

Struct members can’t be declared as protected, protected internal, or private protected since they don’t support inheritance. ( ………. )

**Fields –** a field is any variable that is declared directly in a class or struct. Instance fields are specific to an instance of a type. Static fields belong to the class itself, and is shared among all instances of the class.

A field can be given an initial value when it is declared. It will then be initialized immediately before the constructor for the object instance is called. *A field initializer cannot refer to other instance fields*.l

A readonly field can only be assigned a value during initialization or in a constructor.

**Constants** – constants are immutable values which are known at compile time and do not change during the lifetime of the program. They are declared using the const modifier. Only C# built in types can be declared as const. Use readonly for classes, or structs or arrays that are to be initialized only once at runtime.

**PROPERTIES** – A property is a member that provides a flexible mechanism to read, write, or compute the value of a private field.

They have a get and set accessor that is used to access their backing fields.

**METHODS** – A method is a code block that contains a series of statement.

To pass a value-type instance by reference, use the ref keyword.

Expression body definitions (with syntax => expression ; ) can be used for short methods.

Local functions are private methods of a type that are nested in another member. They can only be called from their containing member. All local variables that are defined in the containing member are accessible in a non-static local function.

A reference return value allows a method to return a reference to a variable, rather than a value back to a caller. The caller can then choose to treat the returned variable as if it were returned by value or by reference.

Parameters can be passed by value or by reference.

Use the keyword var to declare implicitly typed local variables. However, var must be used with anonymous types.

Extension methods enable you to add methods to existing types without creating a new derived type, recompiling or otherwise modifying the original type.

Named Arguments enable you to specify an argument for a parameter by matching the argument with its name rather than its position in the parameter list.

Optional Arguments enable you to omit arguments for some parameters.

**CONSTRUCTORS**

Called when an object is created to instantiate an object.

A static constructor can be used to initialize static members of the type which are otherwise initialized to their respective static values. Static constructors are parameterless. A class can have only one static constructor. Cannot be inherited or overloaded. Is never called directly. It’s called by the CLR automatically.

A private constructor is a special instance constructor, used to prevent object instantiation.

**FINALIZERS**

Finalizers are used to perform any necessary final clean-up when a class instance is being collected by the garbage collector.

**OBJECT INITIALIZERS**

Object initializers let you assign values to any accessible fields or properties of an object at creation time without having to invoke a constructor. We use object initializer syntax.

**COLLECTION INITIALIZERS**

Collection initializers let you specify one or more element initializers when you initialize a collection type that implements IEnumerable and has Add() as an instance method or an extension method.

**NESTED TYPES**

A nested type is a type defined within a class, struct or interface. Are private by default i.e. are accessible only within their containing type.

**PARTIAL CLASSES AND METHODS**

C# allows a class, struct, interface or method to be split over two or more source code files.

To split a class, interface or struct definition, use the partial keyword. It indicates that other parts of the type can be defined in the namespace.

* All parts must use the partial keyword.
* All the parts must have the same accessibility.
* If any part is abstract or sealed, so is the rest of the type.
* If one part inherits a base class, so do all the other parts.

A partial class may contain a partial method. One part of the method defines the signature, the other the implementation.

* They must begin with the partial keyword in their definition.
* They must return void

**ANONYMOUS TYPES**

Use the var keyword.

**INTERFACES**

An interface contains definitions for a group of related functionalities that a non-abstract class or struct must implement.

* Its static methods must have an implementation.
* It cannot contain fields, auto-implemented properties, or property like events.

They can contain methods, properties, events, indexers, static constructors, static fields, constants or operators.

**DELEGATES**

**ARRAYS**

**STRINGS**

**INDEXERS**

**EVENTS**

**GENERICS**

**NAMESPACES**

**UNSAFE CODE AND POINTERS**

**~~XML DOCUMENTATION~~**

**EXCEPTIONS AND EXCEPTION HANDLING**

**FILE SYSTEM AND REGISTRY**

**~~INTEROPERABILITY~~**

LANGUAGE REFERENCE

**TYPES**

**Value types –** a variable of a value type contains an instance of the type, unlike a reference type that contains a reference to an instance.

By default, assignment, passing an argument to a method, and returning a method result, variable values are copied.

All **Structure types** and **enumeration types** are value types.

**Built in types**

**Integral numeric types**

sbyte System.SByte ±128

byte System.Byte +255

short System.Int16 ±32K

ushort System.UInt16 +65k

int System.Int32 ±2.1B

uint System. UInt32 +4B

long System. Int64 ±9QT

ulong System.UInt64 +18QT

decimals – without any prefix

hexadecimal – must have 0x or 0X prefix

binary – must have 0b or 0B prefix

**Floating-point numeric types**

float System.Single ±1038

double System.Double ±10308

decimal System.Decimal ±1028

The float and double types also provide constants that represent not-a-number and infinity values.

**bool – System.Boolean.** A structure type that represents a Boolean value, **true** or **false**.

The default value of the bool is false.

**char – System.Char –** A structure type that represents a Unicode UTF-16 character.

**Enumeration –** enum **-** A type defined by a set of named constants of the underlying integral numeric type.

* They can represent a combination of choices if used as bit flags.

**A structure type** – is a value type that can encapsulate data and related functionality.

They are used to design small data-centric types that provide little or no behavior.

(C# 7.2) readonly structs – a readonly modifier on structs declares a structure type that is immutable.

(C# 8.0) readonly instance members – you can use the readonly modifier to declare that an instance member doesn’t modify the state of a struct.

Limitations

* Structs don’t allow parameterless constructors. Structs already provide an implicit parameterless constructor.
* You can’t initialize an instance field or property at its declaration.
* A constructor of a structure type must initialize all instance fields of the type.
* Cannot inherit or be inherited from, but can implement interfaces.
* You can’t declare a finalizer within a structure type.

Use the ref, out or in modifiers to pass a struct by reference to a method.

Use Struct keyword in a struct constraint to specify that a type parameter is a non-nullable value type.

**Tuple types (C# 7.0) –** tuples feature provides concise syntax to group multiple data elements in a light weight data structure. Eg.

(double, int) tupletype = ( 4.5, 3 ) ;

tupletype.item1; // displays 4.5

(double sum, int count) tupletype = ( 4.5, 3 ) ;

var t = (Sum: 4.5, Count: 3);

**Nullable value types – T? –** Represents all values of its underlying value type T and an additional null value.

It is an instance of the generic System.Nullable<T> structure.

Lifted operators are the unary, binary or any overloaded operators that are supported by the underlying non-nullable types of T. They produce null if any one of their operands are null.

**Reference Types**

Variable of reference types store references to their data objects. (Variables of value types directly contain their data). They include variables of type, class, interface, delegate, record. Some built-in reference types are dynamic, object, string.

**Void**

Use void as the return type of a method (or a local function) to specify that the method doesn’t return a value.

**Var**

Variables that are declared at method scope can have an implicit “type” var. It is strongly typed but the compiler defines the type.

**Built-in types**

**Value types:** bool, bype, sbyte, char, decimal, double, float, int, uint, long, ulong, short, ushort.

**Reference types:** Object, string, dynamic

**Unmanaged types**

Value types, enum types, pointer type, any user-defined struct type that contains fields of unmanaged types only and (in C# 7.3 and earlier) is not a constructed type (a type that includes at least one type argument)

**Default values**

|  |  |
| --- | --- |
| Any reference type | null |
| Any built-in integral numeric type | 0 |
| Any built-in floating point num types | 0 |
| bool | false |
| char | ‘\0’ |

|  |  |
| --- | --- |
| enum | Value produced by expression (E)0, where E is the enum identifier |
| struct | Value produced by setting all value-type field to their default values and all reference-type fields to null. |

The default operator produces the default value of a type. Eg. Int a = default(int);

**KEYWORDS**

**Modifiers**

Access Modifiers **–** public, protected, internal, protected internal, private, private protected

abstract

async

const

event

extern

in (generic modifier)

new

out

override

readonly

sealed

static

unsafe

virtual

volatile

**Method Parameters**

params **–** specifies that this parameter may take a variable number of arguments.

in **–** this parameter is passed by reference but is only read by the called method.

ref **–** this parameter is passed by reference and may be read and written by the called method.

out **–** this parameter is passed by reference and is written by the called method.

**Statement Keywords**

**Selection Statements –** if, else, switch, case

**Iteration Statements –** do, for, foreach, in, while

**Jump Statements –** break, continue, default, goto, return, yield

**Exception Handling –** throw, try-catch, try-finally, try-catch-finally

**Checked and Unchecked –** check, unchecked

**Fixed Statement –** fixed

**Lock statement –** lock

**Namespace Keywords**

namespace **–** used to declare a scope that contains a set of related objects.

using – using statement – defines a scope at the end of which an object will be disposed.

using directive – creates an alias for a namespace, or imports types defined in other namespaces.

using static directive – imports the members of a single class.

**Type-testing keywords**

is **–** an operator that checks if the result of an expression is compatible with a given type.

**Generic Type Constraint Keywords**

new constraint **–** this

where **–** this

**Access Keywords**

base **–** this

this **–** this

**Literal Keywords**

null **–** a literal that represents a null reference.

true and false **–** for boolean types.

default **–** can be used to – specify the default label in the switch statement. – as a default operator or literal to produce the default value of a type.

**Contextual Keywords**

add - Defines a custom event accessor that is invoked when client code subscribes to the event.

async – Indicates that the modified method, lambda expression, or anonymous method is asynchronous.

await – Suspends an async method until an awaited task is completed.

dynamic – Defines a reference type that enables operations in which it occurs to bypass compile-time type checking.

get - Defines an accessor method for a property or an indexer.

global - Alias of the global namespace, which is otherwise unnamed.

partial - Defines partial classes, structs, and interfaces throughout the same compilation unit.

remove - Defines a custom event accessor that is invoked when client code unsubscribes from the event.

set - Defines an accessor method for a property or an indexer.

value - Used to set accessors and to add or remove event handlers.

var - Enables the type of a variable declared at method scope to be determined by the compiler.

when - Specifies a filter condition for a catch block or the case label of a switch statement.

where - Adds constraints to a generic declaration. (See also where).

yield - Used in an iterator block to return a value to the enumerator object or to signal the end of iteration.

**Query Keywords**

from clause **–** this

where clause **–** this

select clause **–** this

group clause **–** this

into **–** this

orderby clause **–** this

join clause **–** this

let clause **–** this

ascending **–** this

descending **–** this

on **–** this

equals **–** this

by **–** this

in **–** this

**OPERATORS AND EXPRESSIONS**

**SPECIAL CHARACTERS**

$ - String interpolation – Identifies a string literal as an interpolated string. An interpolated string is a string literal that may contain interpolated expressions.

An interpolated string is resolved into a result string at runtime.

@ - verbatim identifier – enables C# keywords to be used as identifiers. 2. To indicate that a string literal is to be interpreted verbatim. 3. To enable the compiler to distinguish between attributes in cases of naming confilict.

\_ - digit separator – can be used to separate long numbers. eg 1\_000\_000\_000.

**ATTRIBUTES READ BY THE COMPILER**

**PREPROCESSOR DIRECTIVES**

**COMPILER OPTIONS**

(Skipped)

**COMPILER ERRORS**

C# Programming Language

An object-oriented, type-safe, component-oriented programming language.

It features garbage collection to automatically reclaim memory occupied by unreachable unused objects.

Nullable types guard against variables that don’t refer to allocated objects.

Exception handling provides a structured and extensible approach to error detection and recovery.

Lambda expressions support functional programming techniques.

Language Integrated Query (LINQ) syntax creates a common pattern for working with data from any source.

Language support for asynchronous operations provides syntax for building distributed systems.

A unified type system – all types inherit from a single root object type.

**.NET ARCHITECTURE**

C# programs run on .NET, a virtual execution system called the common language runtime (CLR) which is an implementation of the Common Language Infrastructure.

Source code is compiled into an intermediate language (IL)

# JAVASCRIPT

## BASICS

A literal is a data value that appears directly in a program.

## TYPES AND IDENTIFIERS

Types are divided into two categories: **Primitive** types and **object** types.

**Null** and **undefined** are primitive values, and each is considered to be the sole member of its own special type.

Any value that is not a number, string, boolean, null or undefined is an object.

An object is a collection of properties where each property has a name and a value.

An ordinary JavaScript object is an unordered collection of named values.

An array is an ordered collection of numbered values.

In JS, a function is an object that has executable code associated with it. They are true values and JS programs can treat them like regular objects.

Constructors are functions written to initialize newly created objects.

JS performs automatic garbage collection for memory management.

JS is an object-oriented language. Only objects have methods. (numbers, strings and boolean values use temporary wrapper objects to achieve the illusion of having methods).

JS types can be divided into types **with methods** and **types without**. They can be categorized as **mutable** or **immutable**. Objects and arrays are mutable.

JS converts values liberally from one type to another.

JS variables are untyped.

JS uses lexical scoping.

### Numbers

JS does not make a distinction between integer values and floating-point values. All numbers are represented as floating point values.

**Integers** (base-10, eg 10), **Hexadecimal** (base-16, eg 0x9F) are supported. **Octals** (base-8) is supported by some standards. **Floating-Point Literals** have a decimal point.

Arithmetic operations supported – Addition (+), subtraction (-), Multiplication (\*), Division (/) and Modulo (%, remainder after division), and math functions defined in the math object.

In case of overflow, the result of a number is a special infinity value, either Infinity or -Infinity.

Underflow (a number smaller than the smallest representable number) returns 0 or “negative 0”

* Division by zero is not an error. It returns infinity or -infinity.
* Zero divided by zero produces the special **NOT-A-NUMBER** value, **NaN**.
* Infinity divided by infinity is **NaN** as well.
* NaN does not compare equal to any other value, including itself.
* Negative zero compares equal to positive zero.

### Text

A string is an immutable ordered sequence of 16-bit values, each of which represents a Unicode character. It uses zero-based indexing. String literals use single **or** double quotes.

### Boolean

Represents true or false values. It supports the usual operators &&, ||, !\*.

### Null and Undefined

Null is a language keyword that evaluates to a special value that indicates the absence of a value. Null is regarded as the sole member of its own type.

Undefined value represents a deeper kind of absence. It is the value of variables that have not been initialized and the value of an object or array that does not exist. Undefined is returned by functions with no return value.

Undefined is the sole member of its special type.

The equality operator considers null and undefined to be equal.

### The Global Object

The global object is a regular JavaScript object whose properties are globally defined symbols.

### Wrapper Objects

JavaScript objects are composite values, they are a collection of properties. We refer to the value of a property or methods using the .(dot) notation. Non objects like strings, numbers and booleans are temporarily converted to objects when we try to access properties using a dot notation.

The temporary objects created when you access a property of a non-object are known as wrapper objects. You can’t define new properties on non-objects like strings, Booleans and numbers.

### Immutable Primitive Values and Mutable Object References

* Primitives are immutable.
* Primitives are compared by value.
* Objects are mutable.
* Objects are not compared by value. They are compared by reference. Two object values are the same if and only if they refer to the same underlying object.

### Conversions and Equality

JS notion of equality is very flexible.

=== -> the strict equality operator does not perform conversions when testing for equality.

**Explicit Conversions**. We use the Boolean(), Number(), String() or Object() functions. E.g.

String(false); will convert to “false”.

### Object to Primitive Conversions

All objects convert to true in Object-to-Boolean conversion.

## 6.0 Objects

JS fundamental datatype is the object. An object is a composite value. An object is an unordered collection of properties, each which has a name and a value.

* An object inherits the properties of another object – it’s prototype.
* Objects are dynamic, they can be added and deleted.

Any value that is NOT a string, number, boolean, null or undefined is an object.

Objects are mutable and are manipulated by reference.

A property has a name and a value. The name may be a string. The value may be any JS value, a getter or setter function.

A property has attributes (i.e writable, enumerable and configurable).

An object has object attributes (i.e prototype (or base object), class (a string that categorizes the type of an object), extensible flag (specifies whether new properties may be added to the object).

# WPF

Windows Presentation Foundation (WPF) provides developers with a unified programming model for building line-of-business desktop applications on Windows. Lets you create desktop client applications for Windows

Panels – Containers for laying out visual elements. (e.g. StackPanel, WrapPanel, DockPanel, Grid, Canvas, UniformGrid).

Logical Trees – The logical tree comprises the elements as they are listed in the XAML file (panels, controls…)

Visual Trees – includes the parts that make up the controls and panels.

Window Class – Content of a window can contain text. The text is considered the foreground of the window. And the space around it is called the background.

Brush – Is a graphic that is used to paint an area with its output. Types – SolidColorBrush, GradientBrush (linear & radial), TileBrush (Drawing, Image, Visual).

The Application Class – Is the harness that hosts your program and the infrastructure that allows it to run.

# Windows Presentation Foundation for .NET Core

## Get started

WPF is a UI framework that creates desktop client applications.

**XAML**

XAML is a declarative XML-based language that WPF uses for things such as defining resources or UI elements.

**XAML extensions**

XAML provides syntax for markup extensions

## Fundamentals

### XAML

### Define resources

### Styles and templates

### XAML Language Reference

### Work with data

## XAML

ASP.NET – a lightweight, open source, highly testable presentation framework optimized for use with ASP.NET Core.

**OVERVIEW**

The StartUp Class – This is where 1) – Services required by the app are configured. 2) – The app’s request handling pipeline is defined, as a series of middleware components.

Dependency Injection Services – ASP includes a built-in dependency injection framework that makes configured services available throughout the app.

Middleware – The request handling pipeline is composed as a series of middleware components.

Host – On startup, an ASP.NET core app builds a host which encapsulates the app’s resources (such as the HTTP server implementation, Middleware components, logging, DI services, configuration).

Configuration – The configuration framework gets settings as name-value pairs from an ordered set of configuration providers.

Environments – The execution environments include Development, Staging, and Production.

Logging – Providers of logging include Console, Debug, Event Tracing …

Routing – A route is a URL pattern that is mapped to a handler. (A handler may be a razor page, an action method in an MVC controller, or a middleware)

Make HTTP request – The request

Content Root – The content root is the base path for – the executable hosting app (.exe) – compiled assemblies that make up the app (.dll) – content files (razor files, configuration files, data files db) – the web root.

Web root – The web root is the base path for public, static resource files, such as Stylesheets, JavaScript, Images.

**STARTUP CLASS**

The Startup class configures services and the app’s request pipeline.

**DEPENDENCY INJECTION SERVICES**

ASP.NET supports the DI software design pattern to achieve Inversion of Control between classes and their dependencies.

Code dependencies are problematic because, they cause avoidable code modifications, they cause dependency chains, the implementation will be difficult to isolate and test.

( ------- missing notes ----------- )

**MIDDLEWARE**

Middleware is software that’s assembled into an app pipeline to handle requests and responses. Each component a) – Chooses whether or not to pass the request to the next component in the pipeline. b) – Can perform work before and after the next component.

The request pipeline consists of a sequence of request delegates, called one after the other.

Endpoint – the endpoint middleware executes the filter pipeline for the corresponding app type (either MVC or razor pages).

( ------- missing notes ----------- )

**HOST**

Host – is an object that encapsulates an app’s resources such as: Dependency Injection, Logging, Configuration, IHostedService implementations. The host is typically configured, built, and run by code in the Program class. It is responsible for app startup and lifetime management.

**SERVERS**

The server listens for HTTP requests and surfaces them to the app as a set of request features composed into an HTTPContext.

On Windows -> Kestrel server, IIS Server.

**CONFIGURATION**

Configuration is done using configuration providers. Configuration providers read configuration data from key-value pairs using various sources eg. Settings files, Environment variables, Azure key vault, command line arguments, directory files, .NET objects.

**OPTIONS**

The options pattern uses classes to provide strongly typed access to groups of related settings.

**ENVIRONMENTS**

ASP configures app behavior, based on the runtime environment using an environment variable.

**LOGGING**

Logging providers store logs.

**ROUTING**

Routing is responsible for matching incoming HTTP requests and dispatching those requests to the app’s executable endpoints (endpoints are the app’s units of executable, request-handling code.

Apps can configure routing using controllers, Razor pages, SignalR, gRPC services,

**MAKE HTTP REQUESTS**

An IHTTPClientFactory can be registered and used to configure and create HTTPClient instances in our app.

**STATIC FILES**

Static files such as HTML, CSS, images and JavaScript are assets an ASP app serves directly to clients by default.

**FILTERS**

Filters allow pages to run code before and after a razor page handler is run.

MVC – Views – The view handles the app’s data presentation and user interaction. A view is an HTML template with embedded razor view markup.

Controller – A controller is used to define and group a set of actions.

Action – An action (or an action method) is a method on a controller that handles requests.

Routing – Is defined in routing templates. Controllers use Routing middleware to match the URLs of incoming requests and map them to actions. Route templates are defined in startup code or attributes, describe how URLs are matched to actions and are used to generate URLs for links.

SESSION AND STATE MANAGEMENT

Model Responsibilities – represents the state of the application and any business logic and operations that should be performed by it. The controller creates and populates these ViewModel instances from the model.

View Responsibilities – Views are responsible for presenting content through the user interface. They use the Razor View Engine to embed .NET code in HTML markup.

Controller Responsibilities – represents the components that handle user interaction, work with the model, and ultimately select a view to render. It handles and responds to user input and interaction.

Routing – a URL-mapping component that lets you build applications that have comprehensive and searchable URLs. Routes are defined using a route template syntax that supports route value constraints, defaults and optional values.

* Convention based routing – enables you to globally define the URL formats that your application accepts and how each of those formats maps to a specific action method on a given controller.
* routes.MapRoute(name: "Default", template: "{controller=Home}/{action=Index}/{id?}");
* Attribute routing – enables you to specify routing information by decorating your controllers and actions with attributes that define your application’s routes.

# SQL

Functions

Stored Procedures

Named Instance

# ANDROID APP PROGRAMMING

App components are the essential building blocks of an Android app. Each component is an entry point. There are 4 different types of app components.

* Activities
* Services
* Broadcast receivers
* Content providers

Activities – is the entry point for interacting with the user. It represents a single screen with a user interface.

Services – A service is a general purpose entry point for keeping an app running in the background for all kinds of reasons. It is a component that runs in the background to perform long-running operations or to perform work for remote processes.

* It does not provide a user interface.

Started services – tell the system to keep them running until their work is completed.

Bound services – run because some other app or the system has said that it wants to make use of the service.

Broadcast Receivers – a component that enables the system to deliver events to the app, outside of a regular system flow, allowing the app to respond to system-wide broadcast announcements.

Content Providers – A content provider manages a shared set of app data that you can store in the file system, in a SQL database or in the web, or on any other persistent storage location that your app can access. Through the content provider, other apps can query or modify the data.

Intent – Activities, services and broadcast receivers are activated by an asynchronous message called an intent.

* An intent is an abstract description of an operation to be performed.

Intents bind individual components to each other at runtime.

Content providers are activated by a request from a contentResolver.

Manifest file – An xml file that

1. A place where all components are declared.
2. Identifies any user permission the app requires.
3. Declares the minimum API level required by the App.
4. Declares hardware and software features required by the app.
5. Declares API libraries the app needs to be linked against.

**ACTIVITIES**

Unlike programming paradigms in which apps are launched with a main() method, the Android system initiates code in an activity by invoking special callback methods that correspond to specific stages in its lifecycle.

An app can invoke the activity of another app, rather than the app as an atomic whole.

An activity provides a windows in which the app draws its UI.

Apps may contain multiple screens and therefore Activities, which call each other to perform different actions.

Activities are independent of each other.

Intent filters – provide the ability to launch an activity based not only on an explicit request but also an implicit one.

**Activity LifeCycle**

OnCreate() – Is Fired when your system creates an activity. You must implement this. Use it to initialize essential components of your activity. Call SetContentView()

OnStart() – Called after onCreate(). The activity becomes visible to the user.

OnResume() – Is invoked just before the activity starts interacting with the user.

OnPause() – Called when the activity loses focus and enters a paused state.

OnStop() – Called when the activity is no longer visible to the user.

OnRestart() – Invoked when an activity in the stopped state is about to restart.

OnDestroy() – Invoked before an activity is destroyed.

# JAVA PROGRAMMING LANGUAGE

# SWIFT PROGRAMMING LANGUAGE

**LANGUAGE GUIDE**

1. **THE BASICS**

**Constants and Variables**

**Semicolons**

**Integers**

**Floating-Point Numbers**

**Type Safety and Type Inferencee**

**Numeric Literals**

**Type Aliases**

**Booleans**

**Tuples**

**Optionals**

**Error Handling**

1. **BASIC OPERATORS**

**Terminology**

**Assignment Operator**

**Arithmetic Operators**

**Comparison Operators**

**Ternary Conditional Operator**

**Nil-Coalescing Operator**

**Range Operators**

**Logical Operators**

1. **STRINGS AND CHARACTERS**

**String Literals**

**Initializing an Empty String**

**String Mutability**

**Strings Are Value Types**

**Working with Characters**

**Concatenating Strings and Characters**

**String Interpolation**

**Unicode**

**Counting Characters**

**Accessing and Modifying a String**

**Substrings**

**Comparing Strings**

1. **COLLECTION TYPES**

**Mutability of collections**

**Arrays**

**Sets**

**Performing set operations**

**Dictionaries**

1. **CONTROL FLOW**

**For-In Loops**

**While Loops**

**Control Transfer Statements**

**Early Exit**

**Checking API Availability**

1. **FUNCTIONS**

**Defining and Calling Functions**

**Function Parameters and Return Types**

**Function Argument Labels and Parameter Names**

**Function Types**

**Nested Functions**

1. **CLOSURES**

**Closure Expressions**

**Trailing Closures**

**Capturing Values**

**Closures are Reference Types**

**Escaping Closures**

**Autoclosures**

1. **ENUMERATIONS**

**Enumeration Syntax**

**Matching Enumeration Values with a Switch Statement**

**Iterating over Enumerations Class**

**Associated Values**

**Raw Values**

**Recursive Enumerations**

1. **STRUCTURES AND CLASSES**

**Comparing Structures and Classes**

**Structs and Enums are Value Types**

**Classes are Reference Types**

1. **PROPERTIES**

**Stored Properties**

**Computed Properties**

**Property Observers**

**Property Wrappers**

**Global and Local Variables**

**Type Properties**

1. **METHODS**

**Instance Methods**

**Type Methods**

1. **SUBSCRIPTS**

**Subscript Syntax**

**Subscript Usage**

**Subscript Options**

**Type Subscripts**

1. **INHERITANCE**

**Defining a Base Class**

**Subclassing**

**Overriding**

**Preventing Overrides**

1. **INITIALIZATION**

**Setting Initial Values for Stored Properties**

**Customizing Initialization**

**Default Initializers**

**Initializer Delegation for Value Types**

**Class Inheritance and Initialization**

**Failable Initializers**

**Required Initializers**

**Setting a Default Property Values with a Closure or Function**

1. **DEINITIALIZATION**

**How Deinitialization Works**

**Deinitializers in Action**

1. **OPTIONAL CHAINING**

**Optional Chaining as an Alternative to Forced Unwrapping**

**Defining Model Classes for Optional Chaining**

**Accessing Properties Through Optional Chaining**

**Accessing Subscripts Through Optional Chaining**

**Linking Multiple Levels of Chaining**

**Chaining on Methods with Optional Return Values**

1. **ERROR HANDLING**

**Representing and Throwing Errors**

**Handling Errors**

1. **TYPE CASTING**

**Defining a Class Hierarchy for Type Casting**

**Checking Type**

**Downcasting**

**Type Casting for Any and AnyObject**

1. **NESTED TYPES**

**Nested Types in Action**

**Referring to Nested Types**

**S**

**S**

**S**

**S**

1. **EXTENSIONS**

**Extension Syntax**

**Computed Properties**

**Initializers**

**Methods**

**Subscripts**

**Nested Types**

1. **PROTOCOLS**

**S**

**S**

**S**

**S**

**S**

**S**

1. **GENERICS**

**S**

**S**

**S**

**S**

**S**

**S**

1. **OPAQUE TYPES**

**S**

**S**

**S**

**S**

**S**

**S**

1. **AUTOMATIC REFERENCE COUNTING**

**S**

**S**

**S**

**S**

**S**

**S**

1. **MEMORY SAFETY**

**S**

**S**

**S**

**S**

**S**

**S**

1. **ACCESS CONTROL**

**S**

**S**

**S**

**S**

**S**

**S**

1. **ADVANCED OPERATORS**

**S**

**S**

**S**

**S**

**S**

**S**

**LANGUAGE REFERENCE**

1. **LEXICAL STRUCTURE**

**S**

**S**

**S**

**S**

**S**

**S**

1. **TYPES**

**S**

**S**

**S**

**S**

**S**

**S**

1. **EXPRESSIONS**

**S**

**S**

**S**

**S**

**S**

**S**

1. **STATEMENTS**

**S**

**S**

**S**

**S**

**S**

**S**

1. **DECLARATIONS**

**S**

**S**

**S**

**S**

**S**

**S**

1. **ATTRIBUTES**

**S**

**S**

**S**

**S**

**S**

**S**

1. **PATTERNS**

**S**

**S**

**S**

**S**

**S**

**S**

1. **GENERIC PARAMETERS AND ARGUMENTS**

**S**

**S**

**S**

**S**

**S**

**S**

# WPF

## Application Development

### Application Management Overview

### Windows in WPF

### Navigation

### WPF Add-Ins

### Hosting

### Build and Deploy

## Advanced

### WPF Architecture

### XAML in WPF

### Base Element Classses

### Element Tree and Serialization

### WPF Property System

### Evens

### Input

### Drag and Drop

### Resources

### Documents

### Globalization and Localization

### Layout

### Migration and Interoperability

### Performance

### Threading Model

### WPF Unmanaged API Reference

## Controls

### Controls by Category

### WPF Content Model

### Control Library

### Styles and Templates

### Control Customization

## Graphics and Multimedia

### WPF Graphics Rendering Overview

### Graphics

### 3D Graphics Overview

### Animation Overview

### Multimedia Overview

### Visual Layer Programming

## Security

### WPF Partial trust security

### Platform Security

### Security Engineering

## Graphics in WPF

Transforms and BitmapEffects are applicable to elements derived from UIElement and FrameworkElement classes. These classes define properties that allow you to apply effects to any objects derived from these classes.

**TRANSFORMS**

Transforms allow you to modify the appearance of an element in specific ways. There are three categories:

* Predefined transform classes (Rotate, scale, skew, translate).
* MatrixTransform – allows you to manipulate all the parameters of a transformation individually.
* Transform Group – is a collection of transforms.

You apply a transform to an element by assigning it to one of the element’s two Transform properties – LayoutTransform or RenderTransform.

* If you assign it to the LayoutTransform property, the transformation is performed when WPF is calculating the layout of the element.
* If you assign it to the RenderTransform property, WPF delays applying the transform until the rendering phase of the display.

**The RotateTransform –** rotates the element at a specified angle around a pivot point.

**The TranslateTransform** – moves an element to a different position.

**The SkewTransform** – skews an element at an angle. You can skew the X coordinates, Y coordinates or both.

**The ScaleTransform** – changes the size of an element.

**BITMAPEFFECTS**

This feature allows you to apply one of five, different filters to an element’s appearance. (Beveled, blurred, dropShadow, embossed, outerglow)

**BRUSHES**

Brushes are used to paint an area with a color or graphic.

**SHAPES**

WPF provides six shape elements.

* Five simple shape classes and one called path (which allows you to define arbitrary shapes).
* These are Rectangle, Ellipse, Line, Polyline, Polygon.

**THE GEOMETRY CLASSES**

Allow you to define two-dimensional shapes and paths. Geometry objects are not visible. You use them to specify a shape to paint, to specify a hit test region, or to specify a path along which to animate an object.

* Their function is to define a shape, not to draw one. To render a geometry, you assign it to a path shape.
* The Geometry classes are – LineGeometry, RectangleGeometry, EllipseGeometry, PathGeometry, GeometryGroup, CombinedGeometry, StreamGeometry.

PathMarkup Syntax

**DRAWINGS**

WPF supplies two sets of classes

**The DrawingImage Class –** allows you to use a graphic as an image.

**The DrawingBrush Class –** The DrawingBrush class allows you to package a graphic to be used as a Brush.

## Animation

Animation – in WPF, is the process of having WPF sequentially change the value of a dependency property from one value to another in small increments over a period of time.

* Animation always works on a single dependency property.
* There can be many animations in progress simultaneously.
* Only dependency properties can be animated.
* Animations can’t change the structure of the visual tree.

You must use an animation class that corresponds to the type of that property.

**The AnimationClock Class**

The AnimationClock object keeps track of the amount of time since the beginning of the animation.

The xxxAnimationObject describes the timeline and produces a value of type xxx when queried by its AnimationClock object.

**Storyboards**

The Storyboard class allows you to set up a group of animations that are performed as a set.

* A Storyboard object has a property called children, which is a collection of animation objects. These animations are started when the storyboard is started.
* The animation object has it’s own clock, called a ClockGroup, that is used to synchronize the AnimationClocks of the animations in its Children collection.

**Other Variations on Animation**

There are two other variations on animation.

* Animation with keyframes allows you to set specific values of the property at various points in the progression.
* Animation along a path allow you to move the position of an object along a path.

## Audio and Video

With the SoundPlayer class, your only control over playing a sound is starting it, stopping it. The MddiaPlayer gives you much more control.

# ASP.NET

**Startup class** –

* Services required by the app are configured.
* The app’s request handling pipeline is defined, as a series of middleware components.

**Middleware** – Each component performs operations on an HttpContext and either invokes the next middleware in the pipeline or terminates the request.

**Host** – The host encapsulates all of the app’s resources. There are two different hosts:

* .NET Generic Host
* ASP.NET Core Web Host

**Servers** – (Kestrel) ASP.NET uses an HTTP server implementation to listen for HTTP requests.

**Configuration** – ASP.NET has a configurations framework.

**Routing** – a route is a URL string pattern that is mapped to a handler. The handler is typically a Razor page, an action method in an MVC controller, or a middleware.

fdadf

"EBSCO Information Services (EIS) provides a complete and optimized research solution comprised of e-journals, e-books, and research databases "" all combined with the most powerful discovery service to support the information needs and maximize the research experience of our end-users. Headquartered in Ipswich, MA, EIS employs more than 3,300 people worldwide. We are the leader in our field due to our cutting-edge technology, forward-thinking philosophy, and top-notch workforce. EIS, a division of EBSCO Industries Inc., based in Birmingham, AL, is ranked in the top 200 of the nation's largest, privately held corporations according to Forbes magazine. EBSCO is a company that will motivate you, inspire you, and allow you to grow. We are looking for the best. If you are too, we encourage you to explore our unique opportunities.The mission of a Senior Software Engineer is to design, develop, test, and deploy software using state-of-the-art technologies as part of an Agile team supporting our internal and external customers.This position calls for a Senior Software Engineer who has demonstrated experience in all phases of application development and support in a team environment. This candidate should be a self-starter that can handle multiple priorities simultaneously. Additionally, they should display a pattern of continual learning and a desire to broaden their software development and design skills.Primary ResponsibilitiesDevelop, maintain, extend software components and ensure reliable deployment of new features for a wide range of EBSCO products.Develop software as part of an Agile team in a Scaled Agile (SAFe) environment, working in two-week iterations.Work with product owner, other members of the Agile team, as well as Program Team on the ART to understand business requirements and priorities.Author well-engineered solutions using test-first/test-driven methodologies.Deliver tested, demonstrable, and deployable software within an iteration.Deliver software that is enabled for CI/CD.Help the team make and deliver on its commitments.Contribute to the architecture and design decisions for the product(s).Automate unit and integration tests to achieve high automated test coverage.Assess and incorporate new tools to aid with the development and testing process.Contribute to the continuous improvement of the team.Contribute to the definition and inspection of quality gates within the build and deployment pipeline.Required QualificationsBachelor's Degree in Computer Science, MIS, Computer Engineering or other Information Technology related degree or equivalent experience.5+ years of professional development experience using C# or Java.5+ years of professional web development experience using .NET Application Servers.5+ years of experience in large systems software design and development with hands on experience in RESTful Web Services, HTML, XML/JSON, HTTP, SSL.5+ years' experience of Automated Unit Testing with JUnit, NUnit, etc.Preferred Qualifications2+ years of experience with Mobile Application development.5+ years of experience with DevOps and CI/CD practices.Experience working on an agile team preferred.Familiarity with one or more JavaScript frameworks (React, AngularJS, Backbone.js) and JavaScript libraries.Some experience with Web Services lifecycle (design, build, test, deploy), API versioning and design approaches, tools, inter-operability, and SOA concepts.Familiarity with Microservices and microservice design patterns.Working knowledge of performance testing strategies and techniques.EBSCO Industries, Inc.is an equal opportunity employer and complies with all applicable federal, state, and local fair employment practices laws. EBSCO strictly prohibits and does not tolerate discrimination against employees, applicants, or any other covered persons because of race, color, sex (including pregnancy), age, national origin or ancestry, ethnicity, religion, creed, sexual orientation, gender identity, status as a veteran, and basis of disability or any other federal, state or local protected class. This policy applies to all terms and conditions of employment, including, but not limited to, hiring, training, promotion, discipline, compensation, benefits, and termination of employment.EBSCO complies with the Americans with Disabilities Act (ADA), as amended by the ADA Amendments Act, and all applicable state or local law.EBSCO Industries, Inc.is an equal opportunity employer and complies with all applicable federal, state, and local fair employment practices laws. EBSCO strictly prohibits and does not tolerate discrimination against employees, applicants, or any other covered persons because of race, color, sex (including pregnancy), age, national origin or ancestry, ethnicity, religion, creed, sexual orientation, gender identity, status as a veteran, and basis of disability or any other federal, state or local protected class. This policy applies to all terms and conditions of employment, including, but not limited to, hiring, training, promotion, discipline, compensation, benefits, and termination of employment.EBSCO complies with the Americans with Disabilities Act (ADA), as amended by the ADA Amendments Act, and all applicable state or local law.View EEO PDF" "EBSCO Information Services (EIS) provides a complete and optimized research solution comprised of e-journals, e-books, and research databases "" all combined with the most powerful discovery service to support the information needs and maximize the research experience of our end-users. Headquartered in Ipswich, MA, EIS employs more than 3,300 people worldwide. We are the leader in our field due to our cutting-edge technology, forward-thinking philosophy, and top-notch workforce. EIS, a division of EBSCO Industries Inc., based in Birmingham, AL, is ranked in the top 200 of the nation's largest, privately held corporations according to Forbes magazine. EBSCO is a company that will motivate you, inspire you, and allow you to grow. We are looking for the best. If you are too, we encourage you to explore our unique opportunities.The mission of a Senior Software Engineer is to design, develop, test, and deploy software using state-of-the-art technologies as part of an Agile team supporting our internal and external customers.This position calls for a Senior Software Engineer who has demonstrated experience in all phases of application development and support in a team environment. This candidate should be a self-starter that can handle multiple priorities simultaneously. Additionally, they should display a pattern of continual learning and a desire to broaden their software development and design skills.Primary ResponsibilitiesDevelop, maintain, extend software components and ensure reliable deployment of new features for a wide range of EBSCO products.Develop software as part of an Agile team in a Scaled Agile (SAFe) environment, working in two-week iterations.Work with product owner, other members of the Agile team, as well as Program Team on the ART to understand business requirements and priorities.Author well-engineered solutions using test-first/test-driven methodologies.Deliver tested, demonstrable, and deployable software within an iteration.Deliver software that is enabled for CI/CD.Help the team make and deliver on its commitments.Contribute to the architecture and design decisions for the product(s).Automate unit and integration tests to achieve high automated test coverage.Assess and incorporate new tools to aid with the development and testing process.Contribute to the continuous improvement of the team.Contribute to the definition and inspection of quality gates within the build and deployment pipeline.Required QualificationsBachelor's Degree in Computer Science, MIS, Computer Engineering or other Information Technology related degree or equivalent experience.5+ years of professional development experience using C# or Java.5+ years of professional web development experience using .NET Application Servers.5+ years of experience in large systems software design and development with hands on experience in RESTful Web Services, HTML, XML/JSON, HTTP, SSL.5+ years' experience of Automated Unit Testing with JUnit, NUnit, etc.Preferred Qualifications2+ years of experience with Mobile Application development.5+ years of experience with DevOps and CI/CD practices.Experience working on an agile team preferred.Familiarity with one or more JavaScript frameworks (React, AngularJS, Backbone.js) and JavaScript libraries.Some experience with Web Services lifecycle (design, build, test, deploy), API versioning and design approaches, tools, inter-operability, and SOA concepts.Familiarity with Microservices and microservice design patterns.Working knowledge of performance testing strategies and techniques.EBSCO Industries, Inc.is an equal opportunity employer and complies with all applicable federal, state, and local fair employment practices laws. EBSCO strictly prohibits and does not tolerate discrimination against employees, applicants, or any other covered persons because of race, color, sex (including pregnancy), age, national origin or ancestry, ethnicity, religion, creed, sexual orientation, gender identity, status as a veteran, and basis of disability or any other federal, state or local protected class. This policy applies to all terms and conditions of employment, including, but not limited to, hiring, training, promotion, discipline, compensation, benefits, and termination of employment.EBSCO complies with the Americans with Disabilities Act (ADA), as amended by the ADA Amendments Act, and all applicable state or local law.EBSCO Industries, Inc.is an equal opportunity employer and complies with all applicable federal, state, and local fair employment practices laws. EBSCO strictly prohibits and does not tolerate discrimination against employees, applicants, or any other covered persons because of race, color, sex (including pregnancy), age, national origin or ancestry, ethnicity, religion, creed, sexual orientation, gender identity, status as a veteran, and basis of disability or any other federal, state or local protected class. This policy applies to all terms and conditions of employment, including, but not limited to, hiring, training, promotion, discipline, compensation, benefits, and termination of employment.EBSCO complies with the Americans with Disabilities Act (ADA), as amended by the ADA Amendments Act, and all applicable state or local law.View EEO PDF" "EBSCO Information Services (EIS) provides a complete and optimized research solution comprised of e-journals, e-books, and research databases "" all combined with the most powerful discovery service to support the information needs and maximize the research experience of our end-users. Headquartered in Ipswich, MA, EIS employs more than 3,300 people worldwide. We are the leader in our field due to our cutting-edge technology, forward-thinking philosophy, and top-notch workforce. EIS, a division of EBSCO Industries Inc., based in Birmingham, AL, is ranked in the top 200 of the nation's largest, privately held corporations according to Forbes magazine. EBSCO is a company that will motivate you, inspire you, and allow you to grow. We are looking for the best. If you are too, we encourage you to explore our unique opportunities.The mission of a Senior Software Engineer is to design, develop, test, and deploy software using state-of-the-art technologies as part of an Agile team supporting our internal and external customers.This position calls for a Senior Software Engineer who has demonstrated experience in all phases of application development and support in a team environment. This candidate should be a self-starter that can handle multiple priorities simultaneously. Additionally, they should display a pattern of continual learning and a desire to broaden their software development and design skills.Primary ResponsibilitiesDevelop, maintain, extend software components and ensure reliable deployment of new features for a wide range of EBSCO products.Develop software as part of an Agile team in a Scaled Agile (SAFe) environment, working in two-week iterations.Work with product owner, other members of the Agile team, as well as Program Team on the ART to understand business requirements and priorities.Author well-engineered solutions using test-first/test-driven methodologies.Deliver tested, demonstrable, and deployable software within an iteration.Deliver software that is enabled for CI/CD.Help the team make and deliver on its commitments.Contribute to the architecture and design decisions for the product(s).Automate unit and integration tests to achieve high automated test coverage.Assess and incorporate new tools to aid with the development and testing process.Contribute to the continuous improvement of the team.Contribute to the definition and inspection of quality gates within the build and deployment pipeline.Required QualificationsBachelor's Degree in Computer Science, MIS, Computer Engineering or other Information Technology related degree or equivalent experience.5+ years of professional development experience using C# or Java.5+ years of professional web development experience using .NET Application Servers.5+ years of experience in large systems software design and development with hands on experience in RESTful Web Services, HTML, XML/JSON, HTTP, SSL.5+ years' experience of Automated Unit Testing with JUnit, NUnit, etc.Preferred Qualifications2+ years of experience with Mobile Application development.5+ years of experience with DevOps and CI/CD practices.Experience working on an agile team preferred.Familiarity with one or more JavaScript frameworks (React, AngularJS, Backbone.js) and JavaScript libraries.Some experience with Web Services lifecycle (design, build, test, deploy), API versioning and design approaches, tools, inter-operability, and SOA concepts.Familiarity with Microservices and microservice design patterns.Working knowledge of performance testing strategies and techniques.EBSCO Industries, Inc.is an equal opportunity employer and complies with all applicable federal, state, and local fair employment practices laws. EBSCO strictly prohibits and does not tolerate discrimination against employees, applicants, or any other covered persons because of race, color, sex (including pregnancy), age, national origin or ancestry, ethnicity, religion, creed, sexual orientation, gender identity, status as a veteran, and basis of disability or any other federal, state or local protected class. This policy applies to all terms and conditions of employment, including, but not limited to, hiring, training, promotion, discipline, compensation, benefits, and termination of employment.EBSCO complies with the Americans with Disabilities Act (ADA), as amended by the ADA Amendments Act, and all applicable state or local law.EBSCO Industries, Inc.is an equal opportunity employer and complies with all applicable federal, state, and local fair employment practices laws. EBSCO strictly prohibits and does not tolerate discrimination against employees, applicants, or any other covered persons because of race, color, sex (including pregnancy), age, national origin or ancestry, ethnicity, religion, creed, sexual orientation, gender identity, status as a veteran, and basis of disability or any other federal, state or local protected class. This policy applies to all terms and conditions of employment, including, but not limited to, hiring, training, promotion, discipline, compensation, benefits, and termination of employment.EBSCO complies with the Americans with Disabilities Act (ADA), as amended by the ADA Amendments Act, and all applicable state or local law.View EEO PDF" "EBSCO Information Services (EIS) provides a complete and optimized research solution comprised of e-journals, e-books, and research databases "" all combined with the most powerful discovery service to support the information needs and maximize the research experience of our end-users. Headquartered in Ipswich, MA, EIS employs more than 3,300 people worldwide. We are the leader in our field due to our cutting-edge technology, forward-thinking philosophy, and top-notch workforce. EIS, a division of EBSCO Industries Inc., based in Birmingham, AL, is ranked in the top 200 of the nation's largest, privately held corporations according to Forbes magazine. EBSCO is a company that will motivate you, inspire you, and allow you to grow. We are looking for the best. If you are too, we encourage you to explore our unique opportunities.The mission of a Senior Software Engineer is to design, develop, test, and deploy software using state-of-the-art technologies as part of an Agile team supporting our internal and external customers.This position calls for a Senior Software Engineer who has demonstrated experience in all phases of application development and support in a team environment. This candidate should be a self-starter that can handle multiple priorities simultaneously. Additionally, they should display a pattern of continual learning and a desire to broaden their software development and design skills.Primary ResponsibilitiesDevelop, maintain, extend software components and ensure reliable deployment of new features for a wide range of EBSCO products.Develop software as part of an Agile team in a Scaled Agile (SAFe) environment, working in two-week iterations.Work with product owner, other members of the Agile team, as well as Program Team on the ART to understand business requirements and priorities.Author well-engineered solutions using test-first/test-driven methodologies.Deliver tested, demonstrable, and deployable software within an iteration.Deliver software that is enabled for CI/CD.Help the team make and deliver on its commitments.Contribute to the architecture and design decisions for the product(s).Automate unit and integration tests to achieve high automated test coverage.Assess and incorporate new tools to aid with the development and testing process.Contribute to the continuous improvement of the team.Contribute to the definition and inspection of quality gates within the build and deployment pipeline.Required QualificationsBachelor's Degree in Computer Science, MIS, Computer Engineering or other Information Technology related degree or equivalent experience.5+ years of professional development experience using C# or Java.5+ years of professional web development experience using .NET Application Servers.5+ years of experience in large systems software design and development with hands on experience in RESTful Web Services, HTML, XML/JSON, HTTP, SSL.5+ years' experience of Automated Unit Testing with JUnit, NUnit, etc.Preferred Qualifications2+ years of experience with Mobile Application development.5+ years of experience with DevOps and CI/CD practices.Experience working on an agile team preferred.Familiarity with one or more JavaScript frameworks (React, AngularJS, Backbone.js) and JavaScript libraries.Some experience with Web Services lifecycle (design, build, test, deploy), API versioning and design approaches, tools, inter-operability, and SOA concepts.Familiarity with Microservices and microservice design patterns.Working knowledge of performance testing strategies and techniques.EBSCO Industries, Inc.is an equal opportunity employer and complies with all applicable federal, state, and local fair employment practices laws. EBSCO strictly prohibits and does not tolerate discrimination against employees, applicants, or any other covered persons because of race, color, sex (including pregnancy), age, national origin or ancestry, ethnicity, religion, creed, sexual orientation, gender identity, status as a veteran, and basis of disability or any other federal, state or local protected class. This policy applies to all terms and conditions of employment, including, but not limited to, hiring, training, promotion, discipline, compensation, benefits, and termination of employment.EBSCO complies with the Americans with Disabilities Act (ADA), as amended by the ADA Amendments Act, and all applicable state or local law.EBSCO Industries, Inc.is an equal opportunity employer and complies with all applicable federal, state, and local fair employment practices laws. EBSCO strictly prohibits and does not tolerate discrimination against employees, applicants, or any other covered persons because of race, color, sex (including pregnancy), age, national origin or ancestry, ethnicity, religion, creed, sexual orientation, gender identity, status as a veteran, and basis of disability or any other federal, state or local protected class. This policy applies to all terms and conditions of employment, including, but not limited to, hiring, training, promotion, discipline, compensation, benefits, and termination of employment.EBSCO complies with the Americans with Disabilities Act (ADA), as amended by the ADA Amendments Act, and all applicable state or local law.View EEO PDF"