

(CS411 (Visual programming))

Mid-Term Short Notes.

Max. All Subject + Objective Main points  
Included.

(Wishing you very best of luck.)

Notes prepared by : HALEEMA

Remember me in your prayers.

# (Chapter #01)

- ★ WPF Stand for Windows Representation Foundation.
- ★ GUI Stand for Graphical User Interface.
- ★ C# is a Case Sensitive language.
- ★ C# is Event-Driven programming.
- ★ There are three Classes included in FStreamr;
  - Create • Read • Write.
- Pros and Cons of C#:-
  - ★ D:- Blocking is a huge issue.
  - ★ Nothing is parallel.
  - ★ A:- we do one thing at a time.  
(More Focused Approach)
  - ★ The key Idea is that once you understand the approach, it scales really well to a large number of things you want to Approach.

## Chapter #02:-

### • What is an Event?

It's an occurrence within a particular system or domain. There are two meanings, something that happened and the corresponding detection in computer world. An event captures "something" from the actual occurrence and multiple events may capture "one" occurrence.

### • Probabilistic Events:-

Probabilistic events may or may not relate to an actual occurrence. Every event is represented by an event object. There are various types of events and information in the event describes details of the particular event type.

### • Synchronous and Asynchronous Behavior:-

Synchronous means happening at the same time. Asynchronous is the opposite - not happening at the same time.

EXAMPLE:- Synchronous learning involves students interacting with a

teacher at the same time. Asynchronous learning involves students working outside of a classroom setting at their own pace.

### Some key points:-

- ★ Common Event processing operations include reading, creating, transforming, and deleting events.
- ★ The design, coding and operation of applications that use events, either directly or indirectly is called event-based programming or applications based on event-driven architecture.
- ★ if we don't use event-driven programming or application, we will poll for event.
- ★ wait for single event is blocking operation.

### Chapter # 03

- ★ Events are based on the principle of decoupling.

### Events and Request-Response based Architecture:-

Events have already happened whereas

Requests are asking something to happen.

- \* Events are often one-way.
- \* Events decouple producers and Consumers.
- \* Events can be processed asynchronously.
- \* Events can also be compared to Messages. Messages may contain events. Events may live outside events like in events logs.
- \* An events Channel is a subscription mechanism for Events.
- \* Event-based programming, also called Event-driven architecture (EDA) is an architecture style.
- \* Service-oriented architecture (SOA) is built from request-Response. I move away from monolithic application.

### Event Producer:-

An Event producer is an entity at the edge of an event processing system that introduces events from the system.

## \* Events Consumer:-

An Event Consumer is an entity at the edge of an event processing system that receives Events from the system.

## \* Raw Event:-

A raw event is an event that is introduced into event processing system by an event producer.

## \* Derived Event:-

A Derived Event is an event that is generated as a result of event processing that take place inside an event processing system.

## • Stateless Event Processing:-

An Event processing agent is said to be Stateless if the way it processes one event does not influence the way it processes any subsequent Events.

Date \_\_\_\_\_

## \* Event Stream:-

An event stream is a set of associated events. It is often a temporally totally ordered set.

## \* Homogeneous Event Stream:-

A stream in which all the events must be of the same type is called a Homogeneous Event Stream.

## \* Heterogeneous Event Stream:-

A stream in which the events may be of different types is referred to as a Heterogeneous Event Stream.

## \* Event Attribute:-

An Event attribute is a component of the structure of an event. Each Attribute has a name and a data type. Events can be composed, generalized, and specialized.

\* Event producers can produce Events that are hardware generated, or from human interaction.

## (Chapter #04)

Q:- What is Clean-Room Implementation?

Clean Room Design (also known as Chinese wall technique) is the method of copying a design by reverse Engineering and then recreating it without infringing any of the Copyright and trade Secerete associated with original design.

Clean Room design is useful as a defense against Copyright and trade Secerete. Because it relies on Independent Invention.

Q:- Where do Name for C# comes From?

The Initial Name was 'cool'. which stood for "C-like object Oriented Language".

Q:- What is difference between .Net & C#?

C# is a language and .Net is a platform

## ⇒ TYPES OF C#?

(i) VALUE TYPE~

(ii) REFERENCE TYPE~

### • Value Type:-

Value types are derived from System.ValueType and can always be created, copied and have a default value (int, float, char, System.datetime, enum, struct).

### • Reference Type:-

Variables of Reference Types store reference to their data. With reference types, two variables can reference the same object; therefore operation on one variable can affect the object referenced by the other variable.

The keywords are used to declare reference types:-

- (1) Class (2) Interface (3) delegate  
(4) Record

Built-in Reference types:-

- (1) Dynamic (2) object (3) String.

- Date \_\_\_\_\_
- ⇒ Boxing stores a value types in a reference types.
- ⇒ C# is most often used with some implementation of the Common Language Infrastructure (CLI), which is standardized as ECMA-335 Common Language Infrastructure (CLI).

## (Chapter #05)

### "Hello World Coding"

Using System;  
Class Program

{

Static void Main()

{

Console.WriteLine("Hello world");

}

}

### → MCQS:-

\* Jagged Array or Multi-dimensional Arrays:-

⇒ Jagged is basically Array of Arrays.

⇒ Array Size is an Integer Type value.

- ⇒ It uses Zero-based index.
- ⇒ Switch Statement can work for Booleans, enums, integral types, and Strings.
- ⇒ Elements are accessed using the dot operator.
- ⇒ Object variables are references to the original object not the object themselves.
- ⇒ The "this" pointer in Methods refers to the object on which the Method is called.
- ⇒ Parameter can be 'Ref' parameter which are passed by reference.
- ⇒ "Out" parameter which are used for the Return value.

## (Chapter #06)

⇒ Namespace allow name reuse.

⇒ Console Class can reside in Multiple libraries.

⇒ Class use the word Class followed by name, curly braces and the Class body. It has Constructor (ctor) Create object.

⇒ Ctor does not return any value and it initializes class members.

⇒ To Create New Class in IDE, use the new class wizard.

⇒ Default ctors are written with no arguments when no ctor is written.

⇒ An initializer list can be used to use as alternate constructor.

⇒ Types of Class members in C# are Instance and Static.

⇒ For Every New occurrence there are new instance vars.

## ⇒ Static Ctor:-

Static Ctor exists to initialize class static members. It's called only once. It has no parameters. Destructors (dctors) are called by garbage collector.

⇒ "Public" methods accessed outside class. "Public" classes access outside assembly.

⇒ A class can contain Constructors, Destructors, Fields, Methods, properties, Indexers, Delegates, Events, and Nested classes.

⇒ Inheritance introduces the concept of base classes and derived classes.

⇒ Declaration of inheritance is done as Derived:

⇒ Base:- only single inheritance is supported.

⇒ Derived class is exactly the same as base.

Date \_\_\_\_\_

## Polymorphism:-

- ⇒ Polymorphism means to invoke derived class methods through base class reference during Run-time.
- ⇒ Polymorphism methods are declared with "virtual" keyword.
- ⇒ "Override" keyword is used to provide an Implementation.
- ⇒ Polymorphism needs the signature to be the same.
- ⇒ Because of inheritance, the three derived classes can be treated as the base class.

## (Chapter #07)

- ⇒ Properties allow protected reads & writes to a field of a class. other language requires custom getters and setters.

## (Auto-Implemented Properties:-)

Auto-implemented properties, improve the common-cable. Properties have the same idea as getter and setters.

## => Indexer:-

An Indexer enables your Class to be treated like an Array. Its Implemented using "this" keyword and square Brackets Syntax. It looks like property Implementation.

Indexers Can take any number of parameters. The parameters Can be Integers, Strings, or enums. Additionally they Can be overloaded with different type and number of parameters.

## => "Struct"

27 its a value type, whereas Class is reference-type. Value types hold their value in memory where they are declared, but reference types hold a reference to an object in memory.

27 "Struct" Cant have destructors, Cant have Implementation inheritance.

27 C# built-in types are aliases for .NET Framework types.

=> "Interfaces":

They are declared like a class but have no implementation. There is only declaration of events, indexes, methods and properties. They are inherited by classes which provide the Real Implementation.

Q:- What are interfaces good for if they don't implement functionality?

They are great for putting together plug-n-play like architectures where components can be interchanged at will.

27 Interfaces define a Contract.

27 we can also do polymorphism using interfaces.

Q:- Which keyword is used for correct implementation of an interface in C#?

Interface keyword.

## (Chapter # 08)

### Delegate :-

A delegate is a reference to a method. its like function pointers in some other languages.

Methods are algorithm that operate on data. Sometime a data need special operation.

There are two good solutions:-

- \* one is to implement a Comparator Interface in all types.
- \* The other solution is using delegate.
- \* Events and Delegates work hand in hand.
- \* The delegate may be one of the pre-defined .NET delegate Or one that you declare yourself.

## (Chapter # 09)

### ➔ (Attributes :-)

Attributes add declarative information to your program. They are used for various function/purpose during Run time. They can be used at design time by application development tools. Attributes add metadata to your program.

- \* When your C# program is compiled it creates a file called an Assembly, which is normally an executable or DLL library.
- \* Assemblies are self-describing because they have Metadata.
- \* Attributes are the classes that can be written in C# and used to decorate your code with declarative information.
- \* Attributes are generally applied physically in front of type and type member declaration. They are declared with square brackets. [ ]

- \* Attributes parameters can be either positional parameters or Named parameters.

## Enums (Enumerations) :-

Enums are strongly typed Constants. They are unique type that allow you to assign symbolic names to Integral values. Enum of one type may not be implicitly assigned to an enum of another type.

- \* All assignments between different enums types and Integral types requires an explicit Cast.
- \* C# type, enum inherits the Base Class Library (BCL) type, Enum.
- \* Default underlying type of an enum is "Int".
- \* Valid base types include byte, sbyte, short, int, uint, long, and ulong.
- \* Default value of First member is 0.

## Overloaded Operators:-

Overloaded operators must be static. They must be declared in the class for which the operator is defined. It is required that matching operators are both defined, e.g. `==` and `!=` so that the behaviour is consistent.

~~(Asse)~~

## (Chapter # 10)

### Anonymous Methods:-

Anonymous methods are used with delegates and handlers and events. Anonymous methods result in much less code. An anonymous method is a method without a name. To declare the anonymous method, you use keyword "Delegate".

★ We can create nullable value type by appending a question mark to a type name.

★ Breakpoints allow stopping the program during execution.

# ★ XML (Extensible Markup Language)

XML is widely used for exchanging data. It's readable for both humans and machine. It's a sticker version of HTML. It's made of Tags, Attributes, & values.

## XML Reader:-

There are two methods to Read XML documents;

★ `XmlDocument` ★ `XmlReader`.

`XmlDocument` reads entire document in Memory. Let's you go forward, Backward, Even apply xpath Searches on it.

`XmlReader` is fast, use less Memory and provide one element at a time.

★ Using `XmlDocument` is easier. No order is required. Document element is the Root element and `childNodes` is the set of Children of any node.

★ `XmlNode` is derived from `XmlElement` and contains Name, InnerText,

Innerxml, outerxml, and attributes.

- \* Xpath is a Cross-platform Xml Query language.
- \* There are two Types of XML:-
  - ① Structure (DTD - document Type definition)
  - ② Content (XSD - XML Schema definition).

## (Chapter # 12)

### (WPF) (Windows Presentation Foundation)

- \* It was publicly announced in 2003. WPF4 was released in April 2010.

WPF enables polished user Interfaces which are getting a lot of attention. It enables rapid iterations and major interface changes throughout the development process. It allows to keep user Interface description and implementation separate.

\* GDI was an earlier user Interface library introduced in Windows 1.0 in 1985.

\* XNA Comes with managed library for directx and is great for game development.

### (Highlights of WPF)

- (1) Broad Integration (2D, 3D, Video, Speech libraries etc.)
- (2) Resolution Independence with WPF giving emphasis on vector graphics.
- (3) Hardware acceleration as it is based on Direct3D but it can work using software pipeline also if Direct3D Hardware is not available.
- (4) Declarative programming using extensible Application Markup language (XML)
5. Rich Composition and Customization.

\* Things that Come with WPF 4.0 include Multi-touch support.

- \* Silverlight Comparison is a light-weight version for web. It chooses to WPF approach. First released in 2007 and in April 2010 4th version was released Near WPF 4.
- \* XML is primarily used to describe interfaces in WPF and Silverlight. It is also used to express activities and configurations in workflow Foundation (WF) and windows Communication Foundation (WCF).
- \* XAML is xml with a set of Rules. It is declarative programming language for creating and initializing objects. It's a way to use .net apps.
- \* C# doesn't make sense without .net.

Q: Which keyword is used to define the Macros in C++?

#define

(7)

## (Chapter # 13)

- \* XAML Specification define rules that map .NET namespace, types, properties, and events into XML namespaces, elements, and attributes.
- \* Declaring an XML element in XAML (known as an object element).
- \* Setting an Attributes on an object element is equivalent to setting a property of the same name called a property attributes.
- \* Hooking up an Event Handler of the same name called an event attributes.
- \* XAML can no longer run standalone in the Browser because of the button click Method.
- \* WPF provides type converters for many common data types; Brush, Color, FontWeight, Point, and so on.

### Button Code:- (using SolidColor brush)

```

<Button xmlns="Content"="OK">
  <Button.Background>
    <SolidColorBrush>
      <SolidColorBrush.Color>
        <Color A="255" R="255" G="255" B="255"/>
      </SolidColorBrush.Color>
    </SolidColorBrush>
  </Button.Background>
</Button>

```

- \* Constants in strings cause problems with run time exception not caught at compile time.

## Markup Extensions:-

Markup Extension, like type converters, enables you to extend the expressiveness of XAML.

- \* Markup extensions are invoked from XAML with explicit and consistent syntax.
- \* Using Markup extensions ~~are invoked from XAML~~ is a preferred approach for extending XAML.
- \* An object element can have three types of children.
  - \* A value for a content property.
  - \* Collection items.
  - \* Value that can be type-converted to the object element.
- \* Designated property is the content property.

XAML enables you to add items to the two main types of collections that support indexing: lists and dictionaries.

## (ListBox Code)

```
<listbox> xamlns=  
<listbox.Items>  
<listboxitem Content = "Item 1"/>  
<listboxitem Content = "Item 2"/>  
</listbox.Items>  
</listbox>
```

## (Chapter # 14)

we can also use ~~solid~~ 'SolidColorbrush' code as;

```
<SolidColorbrush> white </SolidColorbrush>
```

OR

```
<SolidColorbrush color = "white"/>
```

OR

```
<Brush> white </Brush>
```

★ XAML work with designated with Marked attributes.   
Classes

## Compiling xaml:-

Compiling xaml involves 3 steps:-

- (1) Converting a Xaml File into a special binary format.
- (2) Embedding the converted content as a binary resource in the assembly being built.
- (3) Performing the plumbing that connect XAML with Procedural code automatically.

## (Chapter # 15)

### BAML:-

BAML is Binary application Markup language. It is just a compressed representation of XAML. There is even BAML Reader available. Earlier there was CAML which stands for Compiled application Markup Language but it is not used now. BAML can be converted back to XAML.

- \* There are different loading mechanisms.
- (i) we can load from Resources.
  - (ii) Identified by original XAML.
  - (iii) Actually Integrated baml loaded.

### (Key Features of XAML 2009:-)

- (1) It includes full generics support using type arguments, dictionary keys of any type.
- (2) Built-in system data types.
- (3) Creating objects with Non-default constructors, getting instance via Factory Methods.
- (4) Event handlers can be Markup extensions returning delegates, define additional members and property/properties.

### (XAML 2006 Keywords Include:-)

#### X:asyncrecords:-

Controls the size of asynchronous XAML-loading chunks.

(9) (X:Class, X:ClassModifier):-

Visibility, public by default.

(X:Code, X:ConnectionId)

Not for public use.

(X:typearguments)

used only with root with X:Class in Xaml 2006.

⇒ We can convert Xaml into C#, often APIs are optimized for Xaml and can look clunky in C#, WPF applications have deep hierarchy small building blocks.

Dispatcher object :-

It is the base class meant for any object that wishes to be accessed only on the thread that created it. Most WPF classes derive from dispatcher object and are therefore thread-unsafe.

Freezable :- It is the base class for objects that can be "Frozen" into a read-only state for performance reasons. Frozen object can never be unfrozen, but you can clone them to create unfrozen copies.

\* Visual is the base class for all objects that have their own 2D visual representation.

## Contentelement:-

Contentelement is a Base Class similar to uielement but for document-related pieces of Content that don't have rendering behavior on their own.

## Frameworkelement:-

Frameworkelement is the base class that adds support for styles, data binding, resources, and a few common mechanism for windows-based control.

FrameworkContent is the analog to Frameworkelement for Content.

## Control:-

Control is the base for Familiar Control such as button, listbox, and statusbar. Control add many properties to its Frameworkelement base class, such as Foreground, Background and fontsize etc.

## (Chapter #16)

## Dependency Properties:-

Dependency properties depends on Multiple providers for determining its value at any time point in time. properties can be set using xaml, directly or in a design tool, without procedural code.

\* Change Notification, Property value inheritance and Support for Multiple providers are Key Feature of Dependency.

\* Dependency properties are represented by System.Windows.dependencyproperty.

\* Visual Studio has a snippet called propdp, that automatically expands into a definition of a dependency property.

\* dependency property support change notification. It's based on metadata at register time.

\* WPF Contains many powerful Mechanisms that independently attempt to set the value of dependency properties. Dependency properties were designed to depend on these providers in a consistent and orderly manner.

(Chapter # 17) and Chp # 18

(Stackpanel Code)

\* Question:- Write XAML Code to Create a "Stackpanel" having four Buttons with dock Setting Top, Center, Bottom, and Stretch:-

```
<Stackpanel>
  <Button horizontalalignment="Left" Backgrounds="
    "Red">Left</Button>
  <Button horizontalalignment="Center" Backgrounds
    ="Orange">Center</Button>
  <Button horizontalalignment="Right" Background="
    "Yellow">Right</Button>
  <Button horizontalalignment="Stretch" Background="
    "Lime">Stretch</Button>
</Stackpanel>
```

Q:- write the XAML Code to create a "DockPanel" having four buttons with dock settings i.e. Top, Bottom, Right and Left.

```
<dockpanel>
1 < Button dockpanel.Dock = "Top" Background = "Red"
  1 (Top) </Button>
< Button dockpanel.Dock = "Left" Background = "Orange" > 2 (Left)
  </Button>
< Button dockpanel.Dock = "Right" Background = "Yellow" > 3 (Right)
  </Button>
< Button dockpanel.Dock = "Bottom" Background = "Lime" > 4
  (Bottom) </Button>
< Button Background = "Aqua" > 5 </Button>
</dockpanel>
```

Simple 'OK' Button code.

```
< Button xmlns = "http://schemas.microsoft.com/winfx
  /2006/xaml/presentation"
  Content = 'OK' />
```

if Height and width of button is given:-

```
< Button xmlns = "http://schemas.microsoft.com/winfx
  /2006/xaml/presentation"
  < Button . Content >
    < Rectangle Height = "40" Width = "40" Fill = "black" />
  </Button . Content >
</Button>
```