Short Answer:

Answer the following questions with complete sentences in your own words. You are encouraged to conduct your own research online or through other methods before answering the questions. If you research online, please consult multiple sources before you write down your answers. You are expected to be able to explain your answers in detail (Provide examples for each question).

1. What is System.IO? What is it used for?

IO(Input and Output) is used to process the input and produce the output (read and write data). Most applications need to process some data and produce some output based on the input. The System.IO namespace contains all the classes required for IO operations.

File class

Provides static methods for the creation, copying, deletion, moving, and opening of a single file, and aids in the creation of FileStream objects.

Use the File class for typical operations such as copying, moving, renaming, creating, opening, deleting, and appending to a single file at a time. You can also use the File class to get and set file attributes or DateTime information related to the creation, access, and writing of a file. If you want to perform operations on multiple files, see Directory.GetFiles or DirectoryInfo.GetFiles.

Many of the File methods return other I/O types when you create or open files. You can use these other types to further manipulate a file. For more information, see specific File members such as OpenText, CreateText, or Create.

Because all File methods are static, it might be more efficient to use a File method rather than a corresponding FileInfo instance method if you want to perform only one action. All File methods require the path to the file that you are manipulating.

The static methods of the File class perform security checks on all methods. If you are going to reuse an object several times, consider using the corresponding instance method of FileInfo instead, because the security check will not always be necessary.

By default, full read/write access to new files is granted to all users.

<https://learn.microsoft.com/en-us/dotnet/api/system.io?view=net-7.0>

1. What is a stream in C#?

● Stream is an abstract class that provides standard methods to transfer bytes to

the source.

● The following classes inherit the Stream class to provide the functionality to

Read/Write bytes from a particular source

Stream

Provides a generic view of a sequence of bytes. This is an abstract class.

Stream is the abstract base class of all streams. A stream is an abstraction of a sequence of bytes, such as a file, an input/output device, an inter-process communication pipe, or a TCP/IP socket. The Stream class and its derived classes provide a generic view of these different types of input and output, and isolate the programmer from the specific details of the operating system and the underlying devices.

Streams involve three fundamental operations:

You can read from streams. Reading is the transfer of data from a stream into a data structure, such as an array of bytes.

You can write to streams. Writing is the transfer of data from a data structure into a stream.

Streams can support seeking. Seeking refers to querying and modifying the current position within a stream. Seek capability depends on the kind of backing store a stream has. For example, network streams have no unified concept of a current position, and therefore typically do not support seeking.

Some of the more commonly used streams that inherit from Stream are FileStream, and MemoryStream.

<https://learn.microsoft.com/en-us/dotnet/api/system.io.stream?view=net-7.0>

1. What is the difference between the File class and FileStream class?

if you are using LARGE files, FileStream is the only way to go - using File will read the entire file into memory. While using FileStream will allow you to read/write parts of the file.

File Class : Provides static methods for the creation, copying, deletion, moving, and opening of files, and aids in the creation of FileStream objects.

FileStream Class : Exposes a Stream around a file, supporting both synchronous and asynchronous read and write operations.

So basically the File Class exposes the "file entry" on disk so that you can handle it as a single unit (create, copy, move, open the file or even read/write the content in one go etc...). A FileStream expose the \*content\* of this file as a stream.

<https://social.msdn.microsoft.com/Forums/en-US/3d1429de-d3cf-4b21-9569-1c02e297b843/difference-between-file-and-filestream?forum=csharplanguage>

1. What does the following code do:

File.WriteAllText(@"C:\ChristmasMusic\Playlist.txt", "This is a playlist for Christmas Hits");

Graphical user interface, application

Description automatically generated

Write “This is a playlist for Christmas Hits” to the file with the file path of C:\ChristmasMusic\Playlist.txt

1. What are serialization and deserialization

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. Its main purpose is to save the state of an object in order to be able to recreate it when needed. The reverse process is called deserialization.

<https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/serialization/>

1. C# exceptions occur in runtime or compile time?

Exception happens at run time

It can be run time exception or compile time exception.

Basically exception or error is point to faults which can be occurred during the program execution or program development. If you don’t handle the exception, then it makes program faulty or give the wrong result.

There are many types of errors in C#.

Compile Time Exceptions:

Compile-time exceptions occur due to syntax errors or incorrect syntax codes. These errors can be found at compile time. These errors are easy to find.

Runtime Exceptions:

Runtime exceptions occur at the time of the actual execution of an application. These exceptions occur due to logical errors such as attempting a number to divide by 0 and so on. These exceptions are difficult to predict or find out. But we can control these errors to some extend.

The exception handling features in C# language helps you deal with any unexpected or exceptional situations that occur when a program is running. Exception handling in C# uses the try, catch, and finally keywords to try actions that may not succeed, to handle failures when you decide that it is reasonable to do so, and to clean up resources afterward. Exceptions can be generated by the common language runtime (CLR), by the .NET Framework or any third-party libraries, or by application code. Exceptions are created by using the throw keyword.

In many cases, an exception may be thrown not by a method that your code has called directly, but by another method further down in the call stack. When this happens, the CLR will unwind the stack, looking for a method with a catch block for the specific exception type, and it will execute the first such catch block that if finds. If it finds no appropriate catch block anywhere in the call stack, it will terminate the process and display a message to the user.

<https://www.c-sharpcorner.com/UploadFile/8a67c0/exception-handling-in-C-Sharp/#:~:text=It%20can%20be%20run%20time,types%20of%20errors%20in%20C%23>.

1. What is a user-deﬁned Exception and why do we need to use a user-deﬁned Exception? How to create one?

if none of the predefined exceptions meets your needs, you can create

your own exception classes by deriving from the Exception class

● How to create your Exception:

○ Define a new class inheriting from the Exception class

○ Override the virtual members that are defined inside the Exception class

based on your need;

○ Throw the custom Exception instance where you need it

User-filtered exception handlers catch and handle exceptions based on

requirements that you defined for the exception. This is useful when one

catch statement(a particular exception object) corresponds to multiple

exceptions;

use the catch statement with the when keyword, only when the condition in

when(condition) is evaluated as true, the catch statement will get executed.

1. Is there anything wrong with the following exception handler as written? Will this code compile?

will not execute

try { }

catch (Exception e) { }

catch (DivideByZeroException a) { }

1. What is the Exception filter?

Try catch block with - User-filtered exception handlers catch and handle exceptions based on requirements that you defined for the exception. This is useful when one catch statement(a particular exception object) corresponds to multiple exceptions; use the catch statement with the when keyword, only when the condition in when(condition) is evaluated as true, the catch statement will get executed.

1. What is a delegate?

● A delegate is a container for holding the reference of a method or function, it’s

described as a “Function Pointer”.

● A delegate can be declared using the delegate keyword, and it can be declared

within a class or a namespace(more common).

● The signature of the method must match the signature of the delegate.

There are 3 steps involved while working with delegates:

1. Declare a delegate (in a class or namespace)

2. Instantiate a delegate

3. Invoke a delegate

1. What is the common usage of delegate?
2. Delegates are often used as:
3. ○ Members of a class
4. ○ Parameters of a function/method
5. What are the built-in generic delegates?

Func Delegate

● Func is a generic delegate included in the System namespace. It has zero or

more input parameters and one out parameter. The last parameter is considered

as an out parameter.

● Func<params\_types..., return\_type> variableName；

Diagram

Description automatically generated with low confidence

Action Delegate

● Action is a delegate type defined in the System namespace. An Action type

delegate is the same as Func delegate except that the Action delegate doesn't

return a value. In other words, an Action delegate can be used with a method

that has a void return type.

● Action<params\_types, ...> variableName;

Graphical user interface

Description automatically generated with medium confidence

Predicate Delegate

● A predicate delegate method must take one input parameter and return bool -

true or false.

Text

Description automatically generated with medium confidence

1. Define a delegate that points to the following methods:

public static string ToNumString (int num)

{

string returnStr = $"Number {num}"; return returnStr;

}

Func<int,string>ToNumStringDelegate = ToNumString;

Console.WriteLine(ToNumStringDelegate());

1. What is a multicast delegate?

Multicast Delegate

● The delegate that points to multiple methods is called a multicast delegate.

● The addition operator adds a function to the invocation list, and the subtraction

operator removes it.

● If a delegate returns a value, then the last assigned target method's value will

be return when a multicast delegate called.

Delegates are often used as:

○ Members of a class

○ Parameters of a function/method

A Multicast Delegate in C# is a delegate that holds the references of more than one handler function. When we invoke the multicast delegate, then all the functions which are referenced by the delegate are going to be invoked. If you want to call multiple methods using a delegate then all the method signatures should be the same.

So, a multicast delegate is just an array of multiple pipelines or multiple delegates. The delegates are going to be invoked in the same order as they are placed in the invocation list. An InvocationList is nothing but an array of delegates or pipelines where each pipeline will dump data into a different method.

In the below example, we have created one delegate whose signature is the same as the two methods i.e. GetArea and GetPerimeter. Then we created the instance of delegate and bind the two methods using the += operator. Similarly, you can use the -= operator to remove a function from the delegate. Once we bind the two methods with the delegate instance and when we call the delegate, both methods are going to be executed. In this case, behind the scene, when we add multiple methods to the delegate, then multiple pipelines are added. In other words, we can now say that the InvocationList now contains two delegates or two pipelines in the same order we add the methods. In this case, the first delegate or pipeline will dump the data into the GetArea method and the second pipeline will dump the data into the GetPerimeter method, and when you will run the application, then you will see that the GetArea method is first executed, and then the GetPerimeter method is going to be executed. And in InvocationList, you will see that we have two pipelines or delegates having the same name in this example.

[Multicast Delegates in C# with Examples - Dot Net Tutorials](https://dotnettutorials.net/lesson/multicast-delegate-csharp/#:~:text=A%20Multicast%20Delegate%20is%20a%20delegate%20that%20holds,all%20the%20method%20signature%20should%20be%20the%20same.)

1. Explain lambda expression, what does it do?

Lambda expression in C# is the shorthand for representing anonymous method.

Parameters => Body expressionGraphical user interface, text, application, email

Description automatically generated

Coding Questions:

Write code in c# to solve the following problems. Please write your own answers. You

are highly encouraged to present more than one way to answer the questions. Please

follow best practices when you write the code so that it is easily readable, maintainable,

and efficient. Clearly state your assumptions if you have any. You may discuss with

others on the questions, but please write your own code.

(For your Collections HW please try to finish them as much as you can, I’ve already extended

the DDL to Next Monday)

1. Read from the Holiday.txt and count the number of occurrences of each word(use

Dictionary).

2. Create a user-defined exception class called NonIntResultException which is

generated when the result of dividing two integers values produces a result with a

fractional component [ i.e the result is not an integer]

• NonIntResultException contains:

• Generates an appropriate message, for example, if the two integers are 7 and 2 . the

resulting exception message would be “7 divided by 2 is not an integer”

3. Create the IntegerArrayMath class with int division method:

A. Loops thru instance field array and attempts to divide each value of the number array

by the corresponding value of denom instance field array. such as number[0]/denom[0]

and number[1]/denom[1],etc

B. If the result of the division is an integer then print out a message indicating the result

of the division such as 8/4 is 2.

C. If the result of the division is not an integer then throw and handle a

NonIntResultException and continue processing the result of the number array

elements.

D. The method should use exception handling and also handle any attempt to divide by

zero(arithmetic exception) the program should display an appropriate message and

then continue processing the rest of the number array elements

E. Assume both arrays are the same length.

Examples:

Input:

Number is [4,8,15] and denom is [2,0,4]

Output:

The resultant output would be: 4/2 is 2

Division by zero is undefined

result 15 divided by 4 is not an integer

4. Use a built-in generic delegate to extract the if-else condition and pass it as a parameter to

the function. We also encourage you to practice passing delegate as method parameters using

your own examples.

Graphical user interface, text, application, chat or text message

Description automatically generated