C# Programming With .NET
(06CS/IS761)

Chapter wise questions and answers appeared in previous year question papers:

UNIT V: Exceptions & Object life Time		Marks & Year Appeared
1	What are bugs, errors and exception? List and explain the core members of System.Exception type. How would you build custom exception. Or List and explain with code, the core members of System.Exception type. Or List and explain the core members of the System.Exception type. How would you build custom exception? Or Mention the methods present in System.Exception base class. Explain TargetSite, StackTrace properties.	June 12 (08m) Dec 10 (10M) Dec 09 (06M) Dec- 11 (10M)

Ans

- **Bugs:**
  - It is an error on the part of the programmer.
  - Being a programmer, if the programmer calls NULL pointer, overflow the bounds of an array, or fail to delete allocated memory (i. e: resulting in memory leak), an bug is created(/generated).
- **Errors:**
  - Errors are caused by the end user of the application.
  - An end user, who enters a malformed string into a textbox that requires a social security number, could generate an error, if you fail to trap this fault in your code base.
- **Exceptions:**
  - Exceptions are run time anomalies, that are difficult, if not impossible, to prevent.
  - Possible exceptions include, attempting to connect to the database that no longer exist. Opening a corrupted file. Connecting to the machine which is offline.

Core members of System.Exception type

System.Exception Property	Meaning in Life
Data	This property retrieves a collection of key/value pairs (represented by an object implementing IDictionary) that provides additional, programmer-defined information about the exception. By default, this collection is empty (e.g., null).
HelpLink	This property returns a URL to a help file or website describing the error in full detail.
InnerException	This read-only property can be used to obtain information about the previous exception(s) that caused the current exception to occur. The previous exception(s) are recorded by passing them into the constructor of the most current exception.
Message	This read-only property returns the textual description of a given error. The error message itself is set as a constructor parameter.
Source	This property returns the name of the assembly that threw the Exception.
StackTrace	This read-only property contains a string that identifies the sequence of calls that triggered the exception. As you might guess, this property is very useful during debugging if you wish to dump the error to an external error log.
TargetSite	This read-only property returns a MethodBase type, which describes numerous details about the method that threw the exception (invoking ToString() will identify the method by name).
InnerException	This read-only property can be used to obtain information about the previous exception(s) that caused the current exception to occur.

	The previous exception(s) are recorded by passing them into the constructor of the most current exception.

//This custome exception describes the details of the car-is-dead condition

```
Public class CarisDeadException : System.Exception
    {
        //This custom exception maintains the name of doomed car.
        private string CarName;
        Private CarIsDeadExeption(){   }
        private CarIsDeadException(string CarName)
           {
                this.CarName = CarName;
           }
//Override the Exception.Message Property
        public override string  Message
        {
                get {
                        string msg = base.Message;
                        if(CarName != null)
                        msg += CarName+"Has brought the farm….";
                        return msg;
                    }
        }
}
```

Here the CarIsDeadException type maintains a private data member that holds the petName of the car, that threw the exception. We had also added two constructors to the class and overridden the Virtual Message property inorder to include the petName of the car in the description.

Throwing this error from within SpeedUp() is straightforward:

```
//Throw the custom CarIsDeadEcxception.
public void SpeedUp(int delta)
{
        //if the car is dead just say so……..
        if(CarIsDead)  {
                //throw car is dead exception…….
                throw new CarIsDeadException(this.PetName);                }
        else
                {……}
}
//Catching the exception is just as easy
try      {……..}
catch (CarIsDeadException e)
{
        Console.WriteLine("Method: {0}",e.TargetSite);
        Console.WriteLine("Message: {0},e.Message");
}
```

2 Ans	Explain the keywords: 1) finally, 2) using.	June 12 (04m)

- Finally: Our try /catch block is also be augmented with an optional finally block.
- The idea is that this block of code will get executed always.

```
 finally
 {
 //This willalways occur. Exception or not.
    buddha.CrankTunes(fals);
 }
```

```
static void Main(string[] args)
{
Console.WriteLine("***** Handling Multiple Exceptions *****\n");
Car myCar = new Car("Rusty", 90);
myCar.CrankTunes(true);
try
{
// Speed up car logic.
}
catch(CarIsDeadException e)
{
// Process CarIsDeadException.
}
catch(ArgumentOutOfRangeException e)
{
// Process ArgumentOutOfRangeException.
}
catch(Exception e)
{
// Process any other Exception.
}
finally
{
// This will always occur. Exception or not.
myCar.CrankTunes(false);
}
Console.WriteLine();
}
```

If you did not include a finally block, the radio would not be turned off if an exception is encountered (which may or may not be problematic). In a more real-world scenario, when you need to dispose of objects, close a file, detach from a database (or whatever), a finally block ensures a location for proper cleanup.

- Using:

3 What is meant by object lifetime? Explain the Garbage Collection optimization process in C#. June 12 (08M)

Or

What is meant by object life time? Describe the role of .NET garbage collection, finalization process and Ad-Hoc destruction method, with examples. Dec- 09 (08M)

Ans

Object Lifetime:

- In C# with .NET programming language, programmers never directly deallocate an object from memory.
- Instead, .NET objects are allocated onto a region of memory termed "*Managed Heap",* where they will be automatically deallocated by the runtime at "some time" in the feature.
- The garbage collector removes an object from the heap when it is *unreachable* by any part of your code base.

Garbage Collection Optimizations:

- To help optimize the process, each object on the heap is assigned to a specific "generation."
- The idea behind generations is simple: the longer an object has existed on the heap, the more likely it is to stay there.
  - For example, the object implementing Main() will be in memory until the program terminates.
  - Conversely, objects that have been recently placed on the heap (such as an object

allocated within a method scope) are likely to be unreachable rather quickly.

- Given these assumptions, each object on the heap belongs to one of the following generations:
  - *Generation 0: Identifies a newly allocated object that has never been marked for collection*
  - *Generation 1: Identifies an object that has survived a garbage collection (i.e., it was marked* for collection, but was not removed due to the fact that the sufficient heap space was acquired)
  - *Generation 2: Identifies an object that has survived more than one sweep of the garbage* collector.
- The garbage collector will investigate all generation 0 objects first.
- If marking and sweeping these objects results in the required amount of free memory, any surviving objects are promoted to generation 1 and cont….

4 Explain how to build custom exceptions in C#, using suitable code. Dec 11 (10M)

```
using System;
using System.IO;

namespace Wrox.ProCSharp.AdvancedCSharp
{
  class MainEntryPoint
  {
    static void Main()
    {
      string fileName;
    Console.Write("Please type in the name of the file " +
      "containing the names of the people to be cold called > ");
    fileName = Console.ReadLine();
    ColdCallFileReader peopleToRing = new ColdCallFileReader();

    try
    {
      peopleToRing.Open(fileName);
      for (int i=0 ; i<peopleToRing.NPeopleToRing; i++)
      {
        peopleToRing.ProcessNextPerson();
      }
      Console.WriteLine("All callers processed correctly");
    }
    catch(FileNotFoundException ex)
    {
      Console.WriteLine("The file {0} does not exist", fileName);
    }
    catch(ColdCallFileFormatException ex)
    {
      Console.WriteLine(
     "The file {0} appears to have been corrupted", fileName);
      Console.WriteLine("Details of problem are: {0}", ex.Message);
      if (ex.InnerException != null)
      {
        Console.WriteLine(
          "Inner exception was: {0}", ex.InnerException.Message);
      }
    }
    catch(Exception ex)
    {
```

```
        Console.WriteLine("Exception occurred:\n" + ex.Message);
      }
      finally
      {
        peopleToRing.Dispose();
      }
      Console.ReadLine();
    }
}
```

5 Explain the process of finalizing objects in .NET environment. Given the members of System.GC and explain their usage, with examples. May-June 10 (09M)

The process of finalizing objects in .NET environment:

- .NET Garbage Collection is a nondeterministic in nature.
  I. e: You are unable to determine exactly when an object will be deallocated from memory.
- But, In some applications you most likely wish to ensure that this resource is released in a timely manner rather then at the time of whim of the .NET Garbage Collection.
- To account such situations, C# class designer is to over ride the virtual System.Object.Finalize() method.
- In case your application make use of unmanaged resources, Can make use of Finalize() method to deallocate the memory blocks.
- The role of finalizer is to ensure that a .NET object can clean up unmanaged resources.

Finalization process takes time, Coz of which it is less practically used.

```
//This Car Overrides ystem.Object.Finalize().
Class FinalizedCar     {
        ~FinalizedCar()     {
                Console.WritlLine("=> Finalizing Car…..");
        }
}
```

6 Write a program in C# to throw and handle the following exceptions in banking application. May-June 10 (11M)

1. MinimumBalanceException: If the amount is less than 1000.
2. ArgumentOutOfRangeException: If the amount deposit is greater than the capacity of an int.Which is an argument to deposit function. Display the details of each exception. Use required members and methods to support the logic.

It is similar to the program No 8. Exception Handling for finding factorial of a number. Define the MinBalance 1000, using #define MinBalance 1000, and follow the procedure of program N0 8.

7 Define a method that would sort an array of integers. Incorporate exception handling mechanism for "index out of bounds" situation. Develop a main program that employs this method to sort a given set of integers. Dec- 10 (10M)

Ans

➢ Exceptional Handling on array overflow And display the auto generated "Index was outside the bounds of the Array" Msg

```
using System;
public class Exceptions
{
   public static int Main(string[] args)
   {
      byte[] myStream = new byte[3];
      try
      {
         for (byte b = 0; b < 10; b++)
         {
            Console.WriteLine("Byte {0}: {1}", b + 1, b);
            myStream[b] = b;
```

```
        }
      }
      catch (Exception e)
      {
         Console.WriteLine("{0}",e.Message);
      }
      return 0;
   }
}
```

8 Write C# application to illustrate handling multiple exceptions. Dec- 10 (06M)

Or

Write C# application to illustrate handling multiple exceptions.

Ans June-July 11 (05M)

```
using System;
namespace SamplePrograms
{
   class Factorial
   {
      public static void Main()
      {
         // Prompt the user to enter their target number to calculate factorial
         Console.WriteLine("Please enter the number for which you want to compute
factorial");
         try
         {
            // Read the input from console and convert to integer data type
            int iTargetNumber = Convert.ToInt32(Console.ReadLine());
            // Factorial of Zero is 1
            if (iTargetNumber == 0)
            {
               Console.WriteLine("Factorial of Zero = 1");
            }
// Compute factorial only for non negative numbers
            else if (iTargetNumber < 0)
            {
             Console.WriteLine("Please enter a positive number greater than 1");
            }
            // If the number is non zero and non negative
            else
            {
               // Declare a variable to hold the factorial result.
               double dFactorialResult = 1;
               // Use for loop to calcualte factorial of the target number
               for (int i = iTargetNumber; i >= 1; i--)
               {
                  dFactorialResult = dFactorialResult * i;
               }
               // Output the result to the console
               Console.WriteLine("Factorial of {0} = {1}", iTargetNumber, dFactorialResult);
            }
         }
catch (FormatException)
         {
            // We get format exception if user enters a word instead of number
            Console.WriteLine("Please enter a valid number", Int32.MaxValue);
         }
```

```
        catch (OverflowException)
        {
            // We get overflow exception if user enters a very big number,
            // which a variable of type Int32 cannot hold
            Console.WriteLine("Please enter a number between 1 and {0}", Int32.MaxValue);
        }
        catch (Exception)
        {
            // Any other unforeseen error
            Console.WriteLine("There is a problem! Please try later");
        }
    }
  }
}
```

9 Explain the different methods of file System.GC type. June july 11 (05M)

- The base class libraries provide a class type named System.GC that allows you to programmatically interact with the garbage collector using a set of static members.

System.GC Member	Meaning in Life
Collect()	Forces the GC to perform a garbage collection. This method has been overloaded to specify a generation to collect, as well as the mode of collection (via the GCCollectionMode enumeration).
GetGeneration()	Returns the generation to which an object currently belongs.
MaxGeneration	Returns the maximum of generations supported on the target system. Under Microsoft's .NET 3.5, there are three possible generations (0, 1, and 2)
SuppressFinalize()	Sets a flag indicating that the specified object should not have its Finalize() method called.
GetTotalMemory( )	Returns the estimated amount of memory (in bytes) currently allocated on the managed heap. The Boolean parameter specifies whether the call should wait for garbage collection to occur before returning.
ReRegisterForFin alize()	Sets the flag indicting that a suppressed object should be registered as finalizable. This assumes that the object was marked as nonfinalizable using SppressFinalize().

10 Define the following keywords with program example: June-july 11 (10M)

1) Try, 2) throw, 3) catch, 4) finally.

11 Why proper ordering of catch blocks is necessary in C#? (05M)

When we are constructing multiple catch blocks for a single try block, we must be aware that when an exception is thrown, it will be processed by the "nearest available" catch block. Thus proper ordering of catch blocks is necessary in C#.

- In the simplest form, a try block has a single corresponding catch block.
- But, in reality, we often run into a situation where the code within try block could trigger numerous possible exceptions.
  - E. g: Car's SpeedUp() method not only throws an exception when you attempt to speed up a doomed automobile, but throws a system- level exception if you send in an invalid parameter like: Argument is any number less than Zero.
- Test for bad parameter…….

```
        Public void SpeedUp(int delta)        {
//Bad Param? Throw system supplied Exception..!
If(delta < 0)
Throw new ArgumentOutofRangeException(Speed must be greater than zero.!);
If(CarIsDead) {
Throw new CarIsDeadException(this.PetName+ "Has brought the farm….!);
}
……..
}
```

- The calling logic will look like this:

```
//Here we are on the lookout of multiple exceptions
try     {
        for(int I = 0;i< 10; i++)
        buddha.SpeedUp(10);
}
catch(CarIsDeadException)   {
        Console.WriteLine("Method: {0}", e.TargetSite);
        Console.WriteLine("Message: {0}",e.Message);
}
catch(ArgumentOutofRangeException) {
        Console.WriteLine("Method: {0}",e.TargetSite);
        Console.WriteLine("Message: {0}",e.Message);
}
// This will catch any other exception beyond CarIsDeadException or
ArgumentOutOfRangeException.
catch(Exception e) {
Console.WriteLine("Method: {0}",e.TargetSite);
        Console.WriteLine("Message: {0}",e.Message);
        }
```

When you are authoring multiple catch blocks, you must be aware that when an exception is thrown, it will be processed by the "first available" catch.