# 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 are objects in C# and why do we need objects?

● An object is the instance of the class, it has states and behaviors as well.

● The state of an object is stored in fields (variables), while methods (functions) display the

object's behavior

● To create an object of a class, specify the class name. followed by the object name, and

use the keyword new.

An object is basically a block of memory that has been allocated and configured according to the blueprint. A program may create many objects of the same class. Objects are also called instances, and they can be stored in either a named variable or in an array or collection.

<https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/object-oriented/objects>

1. Explain the ***static*** keyword in C#. What are static fields and static methods?

● declare a static member, which belongs to the class itself rather than to a

specific object

○ Static

■ Class

■ Compile time

○ Non-static

■ Object

■ Runtime

● Static class

○ If the static keyword is applied to a class, all the members

of the class must be static

Static, in C#, is a keyword that can be used to declare a member of a type so that it is specific to that type. The static modifier can be used with a class, field, method, property, operator, event or constructor.

 a static field (or variable) is shared by all instances of a class. A static method can be overloaded but not overridden. It cannot access non-static members. Because a static method is not called with a reference to an instance, it is faster to invoke a static method on the call stack than an instance method.

<https://www.techopedia.com/definition/4913/static-c#:~:text=Static%2C%20in%20C%23%2C%20is%20a,%2C%20operator%2C%20event%20or%20constructor>.

#### Static Method

A static method is declared with the help of static keyword. Static methods are accessed with the name of the class. A static method can access static and non-static fields, static fields are directly accessed by the static method without class name whereas non-static fields require objects.

#### Static Class

A static class is declared with the help of *static*keyword. A static class can only contain static data members, static methods, and a static constructor. It is not allowed to create objects of the static class. Static classes are [***sealed***](https://www.geeksforgeeks.org/c-sealed-class/), means one cannot inherit a static class from another class.

<https://www.geeksforgeeks.org/static-keyword-in-c-sharp/>

A static class can only include static members. It cannot be instantiated at run time and cannot be inherited. It has its lifetime as that of the application in which it resides.

1. What are extension methods? How to use extension methods? Benefits of extension methods.

Extension methods enable you to "add" methods to existing types without creating a new derived type, recompiling, or otherwise modifying the original type. Extension methods are static methods, but they're called as if they were instance methods on the extended type.

It is a new feature that has been added in C# 3.0 which allows us to add new

methods into a non-static class without editing the code of the class.

● Extension methods must be defined only under the static class.



Extension method is a way to add new functionality into an existing type (both reference and value). LINQ and Dapper are based on Extension methods. We have used extension methods to create custom html helpers in MVC. To create an extension method follow these four steps

Class containing extension method must be static.

Extension method must be static

First parameter to the extension method must be the type to be extended

First parameter must be written after this keyword

<https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/extension-methods>

1. What is Inheritance and how many types of inheritance are supported by C#?

● Inheritance

In inheritance, we define parent and child classes. The child class can inherit all the methods, objects and properties of the parent class. A child class can also have its own methods and specific implementation.

The parent class is also known as a base class and the child class that inherits the base class is also known as derived class.

● Is the ability to derive something specific from something

generic.

● A class can inherit the features of another class and add its own

modification.

● The parent class is the base class and the child class is known

as the subclass or derived class.

● A subclass inherits all the properties and methods of the base

class.

● Aids in the reuse of code.

● C# supports the following inheritance:

○ Single

○ Multi-level

○ Hierarchical

Type of Inheritance ● Single ● Multiple ● Multilevel ● Hierarchical ● Hybrid

Graphical user interface, application, Teams

Description automatically generated

Graphical user interface, application, Teams

Description automatically generated Diagram, Teams

Description automatically generated Teams

Description automatically generated Diagram, Teams

Description automatically generated

1. What is the diamond problem? And how can we resolve the problem?

Multiple and Hybrid cause diamond problems, the ambiguity that can arise as a

consequence of allowing multiple inheritance. The "diamond problem" is an ambiguity that arises when two classes B and C inherit from A, and class D inherits from both B and C. If there is a method in A that B and C have overridden, and D does not override it, then which class of the method does D inherit: that of B, or that of C?

Using interface to solve diamond problem.

<https://www.programmingwithshri.com/2018/09/what-is-diamond-problem-in-c.html#:~:text=The%20%22diamond%20problem%22%20is%20an,B%2C%20or%20that%20of%20C%3F>

1. What is an interface and what is an abstract class? What are the differences between them?

Interface

● Like a class, an interface can have methods, but the methods declared in

the interface are by default abstract (only method signature, no method

body)

○ Interfaces specify what a class or a struct must do rather than how to

do.

○ It defines a contract, any class or struct that implements that contract

must provide an implementation of the members defined in the

interface.

○ If a class implements an interface and does not provide method

bodies for all functions specified in the interface, then the class must

be declared abstract

● Syntax

○ public interface IShape {}

○ public class Triangle : IShape {}

● C# allows multiple inheritance of

Interface

In C#, an interface is used to define the outer abilities of a class. An abstract class is used to define a class’s actual identity, and it is used as the object or the same type.

Method in interface must be public, is comparative slower, not has field ,has only abstract methods

● Abstraction

 Abstraction allows the programmer to display only the necessary details to the world while hiding the others. Abstraction is achieved in C# by using the Abstract class and interface.

A class can be declared as an abstract class by using the “Abstract” keyword. The Abstract class in C# is always the base class in the hierarchy. What makes them different from the other class is that they cannot be instantiated. A C# abstract class needs to be inherited.

Interface vs abstract class:

An abstract class allows you to create functionality that subclasses can implement or override. An interface only allows you to define functionality, not implement it. And whereas a class can extend only one abstract class, it can take advantage of multiple interfaces.

C# abstract class explained

An abstract class is a special type of class that cannot be instantiated. An abstract class is designed to be inherited by subclasses that either implement or override its methods. In other words, abstract classes are either partially implemented or not implemented at all. You can have functionality in your abstract class—the methods in an abstract class can be both abstract and concrete. An abstract class can have constructors—this is one major difference between an abstract class and an interface. You can take advantage of abstract classes to design components and specify some level of common functionality that must be implemented by derived classes.

C# interface explained

An interface is basically a contract—it doesn’t have any implementation. An interface can contain only method declarations; it cannot contain method definitions. Nor can you have any member data in an interface. Whereas an abstract class may contain method definitions, fields, and constructors, an interface may only have declarations of events, methods, and properties. Methods declared in an interface must be implemented by the classes that implement the interface. Note that a class can implement more than one interface but extend only one class. The class that implements the interface should implement all its members. Like an abstract class, an interface cannot be instantiated.

<https://www.infoworld.com/article/2928719/when-to-use-an-abstract-class-vs-interface-in-csharp.html#:~:text=The%20short%20answer%3A%20An%20abstract,take%20advantage%20of%20multiple%20interfaces>.

1. What is Polymorphism?

● Polymorphism in c# is a concept by which we can perform

a single action in different ways.

○ The word "poly" means many and "morphs" means forms. So

polymorphism means many forms.

● There are two types of polymorphism in c#

○ Compile time polymorphism (achieved by method overloading)

Compile time — the instance where the code you entered is

converted to executable● The form is determined at compile time

● Method overloading

○ a class has multiple methods that have the same name but

different parameters

■ Different number of parameters

■ Different data types of parameters

○ Method overloading increases the readability of the

program

○ Runtime polymorphism (achieved by method overriding)

Runtime — the instance where the executable is running

Runtime Polymorphism

● The form is determined at runtime

● Method overriding

○ a method in a subclass has the same name, return type, and

parameters as a method in its base class, then the method in the

subclass is said to override the method in the base class

○ When an overridden method is called through the subclass object,

it will always refer to the version of the method defined by the

subclass. The base class version of the method is hidden.

● The virtual keyword is used to specify the virtual method in the base class,

and the method with the same signature that needs to be overridden in the

derived class is preceded by override keyword.

○ By default, methods are non-virtual. You cannot override a non-virtual method.

○ You cannot use the virtual keyword with the static, private, abstract, or override keyword

Compile-time polymorphism is also known as Static polymorphism. Method overloading is one of the ways in which compile-time polymorphism is achieved. It is known as compile-time polymorphism as the method calling decision is made at the time of compiling.

It is achieved by keeping the method name the same but passing different sets of parameters. In method overloading, the system first checks the parameter used and based on the set of parameters it decides to call the appropriate method.

#2) Dynamic Polymorphism or Runtime Polymorphism

Runtime polymorphism or dynamic polymorphism occurs when both method name and method signature have the same name and parameters. Method overriding is an example of dynamic polymorphism. It allows the user to create an abstract class with partial interface implementation.

Method overriding is achieved using inheritance. To achieve method overriding both the base class and derived class should have the same name and parameter. During compile time the compiler is not able to recognize the overriding method, hence, it doesn’t throw any error. The decision to run a method is taken during runtime.

Dynamic polymorphism is used to implement abstraction. It allows the user to create an abstract class that is used to provide an implementation for an interface when it is inherited by a derived class. The abstract class can contain names/signature of the methods and the derived class can have a more specialized definition for the method.

<https://www.softwaretestinghelp.com/c-sharp/oops-concepts-in-csharp/>

● The base keyword is used to access members of the base class from

within a derived class

● Usage

○ Call a method on the base class that has been overridden by another

method.

○ Use base() to specify which base-class constructor should be called

when creating instances of the derived class.

1. What are the differences between overriding and overloading?

Graphical user interface, text

Description automatically generated

1. What is Encapsulation? How does C# implement it? And why do we need encapsulation?

● Encapsulation is hiding information.

● How

○ Declare fields/variables as private.

○ Provide public get and set methods, to access and update the value of a

private field.

● Why

○ Flexibility — Internal logic changes won’t affect the caller of the method

○ Reusability — Encapsulated code can be used by different callers

○ Maintainability — Operations on encapsulated unit won’t affect other parts

1. What does { get; set; } mean in C#?

The get method returns the value of the variable name.

The set method assigns a value to the name variable. The value keyword represents the value we assign to the property.

Graphical user interface, text, application

Description automatically generated Graphical user interface, text, application

Description automatically generated

1. What is the difference between abstraction and encapsulation?

Website

Description automatically generated with medium confidence

<https://www.geeksforgeeks.org/difference-between-abstraction-and-encapsulation-in-java-with-examples/>

1. What is the ***sealed*** keyword and when do we need it?

we prevent one method from being overridden with sealed keyword.

Graphical user interface, text, application

Description automatically generated

1. Can we use ***this*** keyword in the constructor to invoke other constructors? Why?

A constructor can invoke another constructor in the same object using the this keyword. Like base, this can be used with or without parameters, and any parameters in the constructor are available as parameters to this, or as part of an expression.

This keyword refers to the current object;

Usage:

● this is often used to differentiate between

the constructor parameters and class fields

if they both have the same name.

● this is also used as a modifier of the first

parameter of an extension method

C# provides a powerful keyword known as this keyword and this keyword has many usages. Here we use this keyword to call an overloaded constructor from another constructor.

* When you use this keyword to call a constructor, the constructor should belong to the same class.
* You can also pass parameter in this keyword.
* This keyword always pointing to the members of the same class in which it is used.
* When you use this keyword, it tells the compiler to invoke the default constructor. Or in other words, it means a constructor that does not contain arguments.

<https://www.geeksforgeeks.org/invoking-an-overloaded-constructor-using-this-keyword-in-c-sharp/>

1. What is the difference between ***virtual*** keyword and ***abstract*** keyword?

The main and most important difference between Virtual and Abstract Keywords is that Virtual method/property may or may not be overriden in the derived class. Whereas, in case of abstract keyword, you have to override the method or property, or else the compiler will throw error.

<https://social.technet.microsoft.com/wiki/contents/articles/51300.c-difference-and-similarity-between-virtual-and-abstract-methodproperty-with-example.aspx#:~:text=The%20main%20and%20most%20important,the%20compiler%20will%20throw%20error>.

The virtual keyword is useful in modifying a method, property, indexer, or event. When you have a function defined in a class that you want to be implemented in an inherited class(es), you use virtual functions. The virtual functions could be implemented differently in different inherited class and the call to these functions will be decided at runtime.

The following is a virtual function −

public virtual int area() { }

The abstract keyword in C# is used for abstract classes and abstract functions. An abstract class in C# includes abstract and non-abstract methods.

<https://www.tutorialspoint.com/what-is-the-difference-between-virtual-and-abstract-functions-in-chash>

Graphical user interface, text, website

Description automatically generated

1. What is the Object class in C#? Can you list and explain some methods in the Object class?

The Object class is the base class for all the classes in the .Net Framework. It is present in the System namespace. In C#, the .NET Base Class Library(BCL) has a language-specific alias which is Object class with the fully qualified name as System.Object. Every class in C# is directly or indirectly derived from the Object class. If a class does not extend any other class then it is the direct child class of the Object class and if extends another class then it is indirectly derived. Therefore the Object class methods are available to all C# classes. Hence Object class acts as a root of the inheritance hierarchy in any C# Program. The main purpose of the Object class is to provide low-level services to derived classes.

<https://www.geeksforgeeks.org/c-sharp-object-class/>

Graphical user interface, text, application, email

Description automatically generated

All types in the .NET type system implicitly inherit from Object or a type derived from

it. The common functionality of Object is available to any type.

● The Object class is beneficial if you want to refer any object whose type you don't

know. Notice that the parent class reference variable can refer to the child class

object, known as upcasting

1. What are Boxing and Unboxing?

A Value Type holds the data within its own memory allocation and a Reference Type contains a pointer to another memory location that holds the real data. Reference Type variables are stored in the heap while Value Type variables are stored in the stack.

Stack is used for static memory allocation and Heap for dynamic memory allocation, both stored in the computer's RAM

Boxing is converting a value type to a reference type. Unboxing is to extract the value type of an object.

Ex: int a = 23; Obj b = a;

1. What’s the rule for object casting?

Upcasting is conversion from a derived (child) class to a base (parent) class. It’s like going up the family tree. Downcasting is the opposite: going from a base class to a derived class, down the tree.

Implicit conversions: No special syntax is required because the conversion always succeeds and no data will be lost. Examples include conversions from smaller to larger integral types, and conversions from derived classes to base classes.

1.

// Implicit conversion. A long can

// hold any value an int can hold, and more!

int num = 2147483647;

long bigNum = num;

2. Derived d = new Derived();

// Always OK.

Base b = d;

Explicit conversions (casts): Explicit conversions require a cast expression. Casting is required when information might be lost in the conversion, or when the conversion might not succeed for other reasons. Typical examples include numeric conversion to a type that has less precision or a smaller range, and conversion of a base-class instance to a derived class.

1.

class Test

{

static void Main()

{

double x = 1234.7;

int a;

// Cast double to int.

a = (int)x;

System.Console.WriteLine(a);

}

}

// Output: 1234

2.

// Create a new derived type.

Giraffe g = new Giraffe();

// Implicit conversion to base type is safe.

Animal a = g;

// Explicit conversion is required to cast back

// to derived type. Note: This will compile but will

// throw an exception at run time if the right-side

// object is not in fact a Giraffe.

Giraffe g2 = (Giraffe)a;

Graphical user interface, text, application

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.

1. Write an extension method for string type, to change all the letters in the calling string to upper cases.
2. This is a credit card number validation problem. Credit card numbers follow certain patterns:

* A credit card number must have between 13 and 16 digits.
* It must start with:
  + 4 for Visa cards
  + 5 for Master cards
  + 37 for American Express cards
  + 6 for Discover cards

In 1954, Hans Luhn of IBM proposed an algorithm for validating credit card numbers. The algorithm is useful to determine whether a card number is entered correctly or whether a credit card is scanned correctly by a scanner.

Credit card numbers are generated following this validity check, commonly known as the Luhn check or the Mod 10 check, which can be described as follows (for illustration, consider the card number 4388576018402626):

**Step 1**. Double every second digit from right to left. If doubling of a digit results in a two-digit number, add up the two digits to get a single-digit number.

4388576018402626

2 \* 2 = 4

2 \* 2 = 4

4 \* 2 = 8

1 \* 2 = 2

6 \* 2 = 12 (1 + 2 =3)

5 \* 2 = 10 (1 + 0 =1)

8 \* 2 = 16 (1 + 6 =7)

4 \* 2 = 8

**Step 2**. Now add all single-digit numbers from Step1.

4 + 4 + 8 + 2 + 3 + 1 + 7 + 8 = 37

**Step 3**. Add all digits in the odd places from right to left in the card number 6 + 6 + 0 + 8 + 0 + 7 + 8 + 3 = 38

**Step 4**. Sum the results from Step 2 and Step 3 37 + 38 = 75

**Step 5**. If the result from Step 4 is divisible by 10, the card number is valid; otherwise, it is invalid. For example, the number 4388576018402626 is

invalid, but the number 4388576018410707 is valid.

Your task is to write a program that prompts the user to enter a credit card number as a long integer. Display whether the number is valid or invalid.

Design your program to use the following methods:

/\*\* Return true if the card number is valid \*/

## public static bool isValid(string number)

/\*\* Get the result from Step B \*/

## public static int sumOfDoubleEvenPlace(string number)

/\*\* Return this number if it is a single digit, otherwise, return the sum of the two digits \*/

## public static int getDigit(string number)

/\*\* Return sum of odd-place digits in number \*/

## public static int sumOfOddPlace(string number)

/\*\* Return true if the digit d is a prefix for number \*/

## public static bool prefixMatched(string number, int d)

/\*\* Return the number of digits in d \*/

## public static int getSize(long d)

/\*\* Return the first k number of digits from number. If the number of digits in number is less than k, return the number. \*/

## public static long getPrefix(string number, int k)

1. Design a class named **Person** and its two subclasses named **Student** and

**Employee**. Make **Faculty** and **Staff** subclasses of **Employee**.

* + A Person has a name, address, phone number, and email address.
  + A Student has a class status (freshman, sophomore, junior, or senior). Define the status as a constant. (Here it means to define class status as an enum class.)
  + An Employee has an office, salary, and date hired.
  + A Faculty member has office hours and a rank.
  + A Staff member has a title.

You should also override the ToString() method of the above classes to display the class name and the person’s name.

Your task is to write a test program that creates a Person, Student, Employee, Faculty, and Staff, and invokes their ToString() methods.

1. Guessing Game Project
2. Develop a simple class that represents a number guessing game. The game is played by the program randomly generating a number and the user attempting to guess that number. After each guess, the program will provide a hint to the user identifying the relationship between the number and the guess. If the guess is above the answer then “Too High” is returned, if the guess is below the answer then “Too Low”. Also if the difference between the answer and the guess is less than the difference between the answer and the previous guess, “Getting warmer” is returned. If the difference between the answer and the guess is more

than the difference between the answer and the previous guess, then “Getting Colder” is returned.

1. The program will allow the user to play multiple games. Once a game is complete the user will be prompted to play a new game or quit.
2. Design and build a **GuessingGame** class.
   * 7 instance variables -
     + *answer* - an integer representing the randomly generated number.
     + *generator* – a random Generator object
     + *gameOver* – a Boolean, false if the game is still in progress, true if the game is over.
     + *differential* – an integer representing the difference between a guess and the answer.
     + *max* – maximum value of the number to guess. For example, if the maximum number is 100 then the number to guess would be between 0 and

100. (inclusive)

* + - *maxGuessesAllowed* – the maximum number of guesses the user gets, once this value is passed the game is over.
    - *numGuessesTaken* – an integer that stores the number of guesses taken so far in any game.
  + Constructor and methods -
    - Default Constructor
      * Sets max to zero.
      * Creates the random number generator object.
    - Parameterized Constructor
      * Takes an integer parameter representing the maximum value of the number to guess.
      * Creates the random number generator object.
    - NewGame() method
      * Takes in an integer as a parameter representing the maximum number of guesses and sets the value for *maxGuessesAllowed*.
      * Generates the answer using the random number generator. (0 -

*max*).

* + - * Sets *gameOver* to false.
      * Sets *differential* to the max value.
      * Sets *numGuessTaken* to zero.
    - Guess() method
      * Takes in an integer as a parameter representing a new guess.
      * Compares the new guess with the answer and generates and returns a String representing an appropriate response.
      * The response is based on:
        + The relation of the guess and answer (too high, too low, or correct).
        + The comparison of the difference between the current guess and the answer and the previous guess and the answer. (warmer, colder)
        + Guess out of range error, if the guess is not between 0 and the max number (inclusive) (see sample run below)
        + User has taken too many guesses because *numGuessesTaken* is greater than *maxGuessesAllowed*. If this is the case, set *gameOver* to true.
    - IsGameOver() method - returns the state of game.
      * true if game is over.
      * false if still in progress.
  + Accessor and mutator methods for all instance fields except the Random number generator. Use the Accessor or mutator methods within the other methods of the class rather than directly accessing the instance fields. For example, use mutator methods in the parameterized constructor to modify instance variables.

1. Design and build **GuessingGameTester** class
   * This program will create **GuessingGame** objects and completely test the GuessingGame Class.
   * Hint: The tester shall also provide a nested loop - one loop allows the user to play a new game after the previous game is completed; another loop will prompt the user for a new guess and provide a response based on the guess.

## Example -

**Sample Output – (Note: text in orange represents the user input value)**

Welcome to the Guessing Game Enter the maximum number:

100

Enter the number of guess allowed:

6

Enter your guess, remember it must be between 0 and 100. 50

Too High

Getting Warmer

Enter your guess, remember it must be between 0 and 100. 25

Too High

Getting Warmer

Enter your guess, remember it must be between 0 and 100. 12

Too High

Getting Warmer

Enter your guess, remember it must be between 0 and 100. 6

Too low

Getting Warmer

Enter your guess, remember it must be between 0 and 100. 8

Congratulation

Would you like to play again, enter Y for yes, N for no. Y

Welcome to the Guessing Game Enter the maximum number:

50

Enter the number of guess allowed:

5

Enter your guess, remember it must be between 0 and 50. 60

Guess out of range, The guess must be between 0 and 50. Enter your guess, remember it must be between 0 and 50. 25

Too low

Getting Warmer

Enter your guess, remember it must be between 0 and 50. 48

Too High

Getting Colder

Enter your guess, remember it must be between 0 and 50. 37

Too High

Getting Warmer

Enter your guess, remember it must be between 0 and 50. 36

Congratulation

Would you like to play again, enter Y for yes, N for no. N