1. What are the access modifiers allowed in c#

In C#, there are five access modifiers that control the visibility and accessibility of classes, methods, properties, and other members within a program. These access modifiers determine which parts of your code can access and use those members. The access modifiers in C# are as follows:

A) **Public**: The **public** access modifier allows the member to be accessed from any other code in the same assembly or in other assemblies that reference it. Public members are accessible to all.

B) **Private**: The **private** access modifier restricts the member's accessibility to within the same class or struct. Private members are not accessible from outside the class or struct in which they are declared.

C) **Protected**: The **protected** access modifier allows access to the member within the same class or struct and from any derived class. Protected members are not accessible from outside the class or struct or from unrelated classes.

D) **Internal**: The **internal** access modifier restricts the member's visibility to within the same assembly. Members declared as **internal** are accessible from any code within the same assembly but not from code in other assemblies.

E) **Protected internal**: The **protected internal** access modifier combines the behavior of **protected** and **internal**. It allows access to the member within the same assembly and from any derived class, regardless of whether the derived class is in the same or different assembly.

It's important to note that access modifiers can be applied to classes, structs, interfaces, constructors, fields, properties, methods, and nested types within a class or struct. By default, if no access modifier is specified, members are considered **private** within a class or **internal** within a namespace or assembly.

2. Write a C# program to illustrate the concept of constructor overloading.

```
Using System;

Class Car
{
   Public string Make { get; set; }
   Public string Model {get; set; }
   Public int Year {get; set; }

   // Default constructor
   Public Car()
   {
      Make = "Unknown";
```

```
        Model = "Unknown";
        Year = 0;
    }

    // Constructor with two parameters
    Public Car (string make, string model)
    {
        Make = make;
        Model = model;
        Year = 0;
    }

    // Constructor with three parameters
    Public Car (string make, string model, int year)
    {
        Make = make;
        Model = model;
        Year = year;
    }

    Public void DisplayInfo()
    {
        Console.WriteLine($"Make: {Make}, Model: {Model}, Year: {Year}");
    }
}

class Program
{
    static void Main()
    {
        Car car1 = new Car();
        Car car2 = new Car("Ford", "Mustang");
        Car car3 = new Car("Tesla", "Model S", 2022);

        car1.DisplayInfo(); // Output: Make: Unknown, Model: Unknown, Year: 0
        car2.DisplayInfo(); // Output: Make: Ford, Model: Mustang, Year: 0
        car3.DisplayInfo(); // Output: Make: Tesla, Model: Model S, Year: 2022
    }
}
```

3**. Explain interfaces and write a C# program for extending and combining interfaces**

In C#, an interface defines a contract that specifies a set of members (methods, properties, events, and indexers) that a class must implement. It defines the behavior or capabilities that a class should have without providing any implementation details. In other words, an interface defines what a class can do, but not how it does it.

Interfaces play a crucial role in achieving abstraction and implementing polymorphism in C#. They allow you to define common behavior that multiple classes can share, regardless of their underlying implementation. A class can implement one or more interfaces, thereby inheriting the contract defined by each interface and providing the necessary implementation for the interface members.

To extend or combine interfaces, you can simply specify multiple interfaces that a class should implement, separated by commas. The class then needs to provide implementation for all the members defined in each interface.

Here's an example program that demonstrates extending and combining interfaces:

```
using System;

// Define two interfaces
interface IWalkable
{
   void Walk();
}

interface IFlyable
{
   void Fly();
}

// Implement the interfaces in a class
class Bird : IWalkable, IFlyable
{
   public void Walk()
   {
      Console.WriteLine("Bird is walking.");
   }

   public void Fly()
   {
      Console.WriteLine("Bird is flying.");
   }
}

class Program
{
   static void Main()
   {
      Bird bird = new Bird();
      bird.Walk(); // Output: Bird is walking.
      bird.Fly();  // Output: Bird is flying.
   }
}
```

4. What are delegates? How are they useful? Explain.

In C#, a delegate is a type that represents a reference to a method. It allows you to treat methods as first-class entities, meaning you can pass methods as arguments to other methods, store them in variables, and invoke them later. Delegates provide a way to implement callback mechanisms, event handling, and function pointers.

Delegates are particularly useful in scenarios where you want to decouple the calling code from the actual method implementation. They enable you to write flexible and extensible code by allowing different methods to be dynamically assigned and invoked based on specific conditions or events.

Here are a few use cases and benefits of delegates in C#:

1. Callback Mechanisms: Delegates enable you to implement callback mechanisms, where you can pass a method as an argument to another method. The receiving method can then invoke the callback method at a specific point, allowing you to customize behavior and provide extensibility. This is commonly used in event handling scenarios, such as button click events or asynchronous programming.
2. Event Handling: Delegates are extensively used for implementing event-driven programming. Events are based on the publisher-subscriber pattern, where one component raises an event, and multiple subscribers (delegate methods) can register to be notified when the event occurs. Delegates provide the means to encapsulate and invoke these subscriber methods.
3. Decoupling Code: Delegates help decouple code by separating the implementation details from the code that calls the methods. The caller doesn't need to know the specific implementation of the delegate method; it only needs to know the delegate's signature. This allows for more modular and maintainable code, as you can easily swap or add different implementations without affecting the calling code.
4. Dynamic Method Invocation: Delegates provide a way to invoke methods dynamically at runtime. This is useful when you want to determine which method to call based on certain conditions or user input. By assigning different methods to a delegate variable and invoking it, you can dynamically change the behavior of your program.

Here's a simple example to illustrate the usage of delegates:

```
using System;

// Declare a delegate type
delegate void PrintDelegate(string message);

class Program
{
   static void Main()
   {
```

```
        // Create an instance of the delegate and assign a method to it
        PrintDelegate printDelegate = Console.WriteLine;

        // Invoke the delegate
        printDelegate("Hello, delegates!");

        // Assign a different method to the delegate
        printDelegate = Console.WriteLine;
        printDelegate("Delegate example");

        // Assign a custom method to the delegate
        printDelegate = CustomPrintMethod;
        printDelegate("Custom method");
    }

    static void CustomPrintMethod(string message)
    {
        Console.WriteLine($"Custom: {message}");
    }
}
```

5. **What is LINQ? What are the important features of LINQ**?

LINQ (Language Integrated Query) is a component of the .NET framework that provides a consistent query syntax to access and manipulate data from different sources, such as collections, databases, XML documents, and more. It allows developers to write queries using a common syntax regardless of the underlying data source, enabling a seamless integration of querying capabilities into the programming language.

The important features of LINQ include:

1. Query syntax: LINQ provides a query syntax that resembles SQL (Structured Query Language), making it easier for developers familiar with SQL to write queries against various data sources. It allows you to express complex data operations in a concise and readable manner.
2. Integration with programming languages: LINQ is integrated into programming languages like C# and Visual Basic, which means you can write LINQ queries directly in your code without the need for separate query languages or tools.
3. Type safety: LINQ is statically typed, meaning that the queries are checked for correctness at compile-time. This ensures type safety and helps catch errors early in the development process.
4. Provider model: LINQ provides a flexible provider model that allows you to extend its capabilities to work with different data sources. There are built-in providers for querying in-

memory objects (LINQ to Objects), databases (LINQ to SQL, Entity Framework), XML (LINQ to XML), and more.

5. Deferred execution: LINQ uses deferred execution, which means that queries are not executed immediately when they are defined. Instead, the execution is deferred until the results are actually enumerated or accessed. This allows for more efficient query execution and optimization.
6. Composition: LINQ queries can be composed together, allowing you to combine multiple queries into a single query. This enables complex data manipulations and transformations in a declarative and readable way.
7. Expression trees: LINQ queries can be represented as expression trees, which are data structures that represent the structure and logic of a query in a language-agnostic way. Expression trees can be inspected and manipulated at runtime, providing a powerful mechanism for building dynamic queries.

Overall, LINQ provides a unified querying syntax and a set of powerful features that simplify and streamline data access and manipulation in .NET applications, making it easier to work with different data sources and perform complex data operations.

6. **Describe interface and list down all the features of interface.**

An interface refers to a connection or interaction between two or more systems, devices, or entities. It allows them to communicate and exchange information. In the context of computer systems and software, an interface serves as a means for users to interact with a program or system.

A user interface (UI) specifically relates to the presentation and control of information on a digital device, such as a computer, smartphone, or tablet. It enables users to interact with software applications and perform tasks effectively. User interfaces can be categorized into different types, such as graphical user interfaces (GUI), command-line interfaces (CLI), and voice user interfaces (VUI). Here are some common features and components found in user interfaces:

1. Visual Elements: UIs typically consist of visual components that provide a visual representation of information and options. These elements include windows, menus, buttons, icons, images, and text.
2. Input Controls: These components enable users to input data or make selections. Examples include text fields, checkboxes, radio buttons, dropdown lists, sliders, and date pickers.
3. Navigation: User interfaces often include navigation elements to help users move between different sections or screens of an application. This can include menus, tabs, breadcrumbs, hyperlinks, and search bars.
4. Feedback and Notifications: Interfaces provide feedback to users to indicate the outcome of their actions or the status of a process. This can include error messages, success notifications, progress bars, tooltips, and confirmation prompts.
5. Dialogs and Modals: Interfaces may utilize dialogs or modals to display additional information, prompt for user input, or confirm certain actions. Dialogs typically appear as separate windows or overlays on top of the main interface.

6. Layout and Structure: UIs need to be organized in a logical and intuitive manner. This involves arranging visual elements, grouping related controls, using grids, and employing consistent placement of components.
7. Responsiveness: A good user interface should respond promptly to user actions and provide real-time updates. This includes immediate feedback when interacting with elements, smooth animations, and quick loading times.
8. Accessibility: Interfaces should be designed to be inclusive and accessible to users with disabilities. This involves considering factors such as screen readers, keyboard navigation, color contrast, and alternative text for images.
9. Customizability: Some interfaces allow users to customize certain aspects according to their preferences. This can include options to change themes, layouts, font sizes, or shortcut keys.
10. Help and Documentation: User interfaces may offer contextual help, tooltips, or a dedicated help section to assist users in understanding the application's functionality and features.

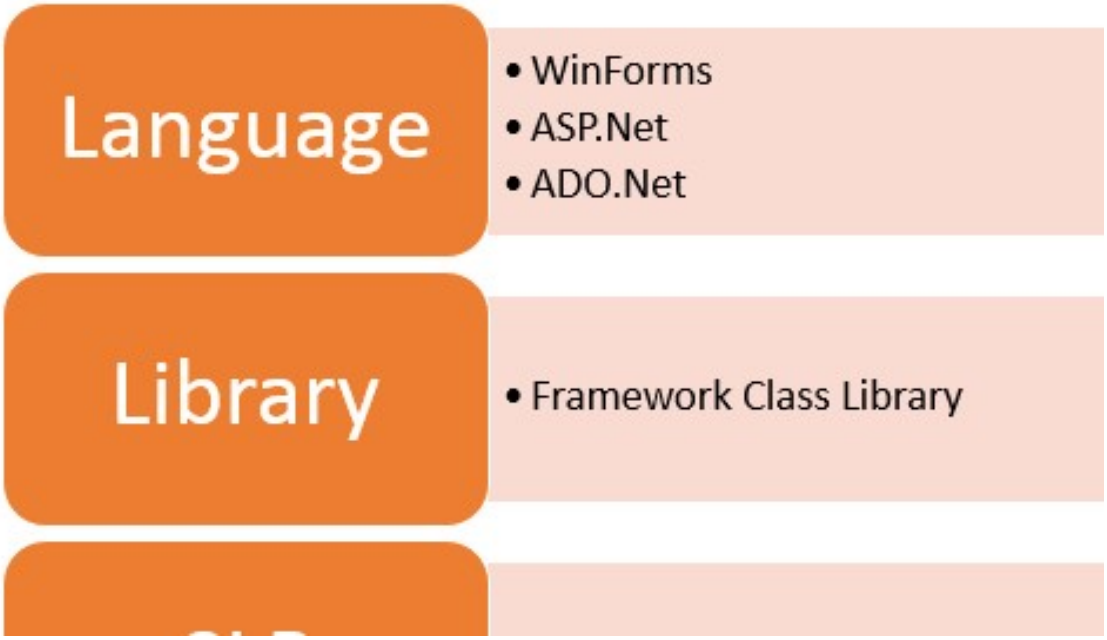
    These features are not exhaustive, as user interfaces can vary greatly depending on the application's purpose, platform, and target audience. The design and functionality of an interface should aim to provide a user-friendly experience that is intuitive, efficient, and visually appealing.

7. Explain .net framework architecture with diagram

The .NET Framework is a software framework developed by Microsoft that provides a comprehensive and consistent programming model for building and running applications. The framework includes a large library of pre-built components and a runtime environment that manages the execution of applications. Here's an overview of the .NET Framework architecture along with a simplified diagram:

.Net Framework Architecture is a programming model for the .Net platform that provides an execution environment and integration with various programming languages for simple development and deployment of various Windows and desktop applications. It consists of class libraries and reusable components.

The basic architecture of the .Net framework is as shown below.



1. Common Language Runtime (CLR): At the core of the .NET Framework architecture is the Common Language Runtime (CLR). The CLR provides the runtime environment for executing .NET applications. It includes a variety of services such as memory management, garbage collection, exception handling, security, and thread management. The CLR also contains the Just-In-Time (JIT) compiler, which translates the Intermediate Language (IL) code into native machine code at runtime.
2. Class Library: The .NET Framework Class Library is a collection of reusable types, classes, and APIs that developers can use to build applications. It provides a wide range of functionality for tasks such as file I/O, networking, database access, user interface development, cryptography, and more. The Class Library is organized into namespaces, which group related classes together.
3. Common Type System (CTS): The Common Type System defines a set of rules that all .NET languages must follow, ensuring type safety and interoperability. It provides a common set of data types and rules for defining and using classes, interfaces, structures, and other constructs. This allows objects created in one .NET language to be used seamlessly by another .NET language.
4. Common Intermediate Language (CIL): When you write code in a .NET language like C# or Visual Basic, it gets compiled into an intermediate language called Common Intermediate Language (CIL) or byte code. CIL is a platform-agnostic representation of the code that can be executed by the CLR. This enables cross-language compatibility and allows the same code to run on different operating systems and hardware architectures.

8 **explain constructor and destructor with example.**

In C#, a constructor is a special method that is called automatically when an object of a class is created. It is used to initialize the object's state and perform any necessary setup

operations. A destructor, on the other hand, is used to clean up resources and perform finalization tasks before an object is destroyed or garbage collected.

Here's an example that demonstrates the use of a constructor and a destructor in C#:

```
using System;


public class MyClass
{
    private string name;

    // Constructor
    public MyClass(string name)
    {
        this.name = name;
        Console.WriteLine("Constructor called. Name: " + name);
    }

    // Destructor (finalizer)
    ~MyClass()
    {
        Console.WriteLine("Destructor called. Name: " + name);
    }
}

public class Program
{
```

```
    public static void Main(string[] args)
    {
        // Creating objects of MyClass
        MyClass obj1 = new MyClass("Object 1");
        MyClass obj2 = new MyClass("Object 2");

        // Performing some operations with the objects

        // ...

        // The objects will be automatically destroyed
    }
}
```

In the above example, the **MyClass** has a constructor that takes a **name** parameter. When objects **obj1** and **obj2** are created using the constructor, the constructor is called, and the provided names are displayed. This is useful for initializing the object's internal state or performing any necessary setup tasks.

The class also has a destructor defined using the **~** symbol followed by the class name. In this case, the destructor simply displays a message with the name of the object being destroyed.

In the **Main** method, two objects of **MyClass** are created, and some operations are performed with them (which are omitted in the example). When the **Main** method ends, the objects are no longer accessible, and they are automatically destroyed. The destructors are called, and messages indicating the destruction of the objects are displayed.

It's worth noting that in C#, destructors are automatically called by the garbage collector when an object is being finalized for garbage collection. It's not guaranteed when exactly the destructor will be called, as it depends on the garbage collection process.

9 Define CLR. How CLR works?

CLR stands for Common Language Runtime. It is a component of the .NET framework developed by Microsoft that provides the necessary infrastructure for executing and managing applications written in various programming languages.

The CLR serves as the execution engine for .NET applications. It provides several key functionalities, including memory management, exception handling, type safety, security enforcement, and code execution. Here's how the CLR works:

1. Compilation: When you write code in a .NET-supported language such as C#, VB.NET, or F#, it is compiled into an intermediate language (IL) code called Common Intermediate Language (CIL). The CIL code is platform-agnostic and can run on any system with the CLR.
2. Just-in-time (JIT) compilation: When an application is executed, the CLR's JIT compiler converts the IL code into machine code specific to the underlying hardware and operating system. This process is known as JIT compilation and occurs at runtime. The JIT compiler optimizes the code for performance by analyzing the execution context.
3. Memory management: The CLR manages memory allocation and deallocation for .NET applications using a technique called garbage collection. It automatically identifies and reclaims memory that is no longer in use, freeing developers from manual memory management tasks. Garbage collection helps prevent memory leaks and ensures efficient memory utilization.
4. Exception handling: The CLR provides a robust exception handling mechanism that allows developers to handle and propagate exceptions in a structured manner. When an exception occurs, the CLR unwinds the call stack, searching for an appropriate exception handler. This ensures that exceptions are properly caught and processed, preventing program crashes.
5. Type safety and verification: The CLR enforces type safety by performing rigorous verification of the IL code before JIT compilation. It ensures that type-related operations are valid, preventing type mismatch errors and other security vulnerabilities.
6. Security enforcement: The CLR enforces various security measures to ensure safe execution of .NET applications. It employs a security model that restricts certain operations, such as accessing system resources, to prevent unauthorized actions by code. The CLR also supports code access security and sandboxing to isolate potentially untrusted code.
7. Code execution: Once the IL code is compiled into native machine code, the CLR executes the application. It manages the execution environment, provides thread synchronization and coordination, and offers other runtime services required by the application.

The CLR plays a crucial role in making .NET a powerful and versatile platform for developing applications that can run on different operating systems and architectures, promoting language interoperability and easing development and deployment processes.

11. **How events are handled through delegates? Explain with example.**

    In programming languages such as C# and some other object-oriented languages, events are handled through delegates. A delegate is a type that represents references to methods with a particular signature. It allows methods to be treated as first-class objects and can be used to subscribe and unsubscribe from events.

    Here's an example to illustrate how events are handled through delegates in C#:

Let's say we have a class called **Button** that represents a button control in a graphical user interface. We want to define an event called **Click** that will be triggered when the button is clicked.

```
public class Button
{
    // Define a delegate type for the event handler
    public delegate void ClickEventHandler();

    // Define an event of the delegate type
    public event ClickEventHandler Click;

    // Method to simulate the button click
    public void OnClick()
    {
        // Check if there are any subscribers to the Click event
        if (Click != null)
        {
            // Invoke the event, which will call all subscribed methods
            Click.Invoke();
        }
    }
}
```

In the code above, we declare a delegate type **ClickEventHandler** that represents the signature of the methods that can handle the **Click** event. Then, we define an event **Click** of the **ClickEventHandler** delegate type.

To handle the event, other parts of the code can subscribe to the event by providing a method that matches the delegate's signature. Here's an example of how an event handler method can be defined and subscribed to the **Click** event:

```
public class Program

{

   static void Main()

   {

      Button button = new Button();

      button.Click += Button_Click; // Subscribe to the Click event



      // Simulate a button click

      button.OnClick();

   }



   // Event handler method

   static void Button_Click()

   {

      Console.WriteLine("Button clicked!");

   }

}
```

13. Write a c# code to find highest among three numbers

Sure! Here's a C# code snippet that finds the highest among three numbers:

```
using System;
```

```
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Enter three numbers:");

        Console.Write("Number 1: ");
        int num1 = Convert.ToInt32(Console.ReadLine());

        Console.Write("Number 2: ");
        int num2 = Convert.ToInt32(Console.ReadLine());

        Console.Write("Number 3: ");
        int num3 = Convert.ToInt32(Console.ReadLine());

        int max = GetMax(num1, num2, num3);

        Console.WriteLine("The highest number is: " + max);
    }

    static int GetMax(int num1, int num2, int num3)
    {
        int max = num1;
```

```
        if (num2 > max)
        {
            max = num2;
        }

        if (num3 > max)
        {
            max = num3;
        }

        return max;
    }
}
```

In this code, we prompt the user to enter three numbers, and then we call the **GetMax** function to find the highest among those numbers. The **GetMax** function compares the numbers and updates the **max** variable accordingly. Finally, we display the highest number to the user.

14 Write the steps required for established database connectivity using ado.net.

1. Import the necessary namespaces:

using System.Data;

using System.Data.SqlClient; // Replace with the appropriate database provider namespace

2. Define the connection string:

string connectionString = "Data Source=server_name;Initial Catalog=database_name;User ID=username;Password=password";

Replace **server_name**, **database_name**, **username**, and **password** with the appropriate values for your database.

3. Create a connection object:

SqlConnection connection = new SqlConnection(connectionString);

4. Open the database connection:

connection.Open();

5. Prepare a SQL query or stored procedure:

string query = "SELECT * FROM table_name"; // Replace with your own query

6. Create a command object and associate it with the connection and query:

SqlCommand command = new SqlCommand(query, connection);

7. (Optional) Set command parameters if you're using parameterized queries:

command.Parameters.AddWithValue("@parameter_name", parameter_value);

8. Execute the command and retrieve the data:

- For queries that return data:

SqlDataReader reader = command.ExecuteReader();

while (reader.Read())

{

// Access data using reader methods, e.g., reader.GetString(column_index)

}

reader.Close();

For queries that don't return data (e.g., INSERT, UPDATE, DELETE):

int rowsAffected = command.ExecuteNonQuery();

9.Close the connection:

connection.Close();

10. Dispose of resources (optional but recommended):

command.Dispose();

```
connection.Dispose();
```

These steps provide a basic structure for establishing database connectivity using ADO.NET. Remember to handle exceptions and implement error checking as appropriate to ensure reliable database interactions.

15. What is interface? How is it different from the class? Explain with example.

In object-oriented programming, an interface is a programming construct that defines a contract or a set of methods that a class must implement. It specifies the methods that a class should provide without specifying how those methods are implemented. An interface can also define properties, events, and indexers. In C#, an interface is declared using the **interface** keyword.

On the other hand, a class is a blueprint or a template that defines the data and behavior of objects. It is a fundamental building block in object-oriented programming and provides a way to create objects based on the defined blueprint. A class can have fields, properties, methods, events, and constructors.

Here's an example to illustrate the difference between an interface and a class in C#:

```
// Interface declaration

public interface IShape

{

   double CalculateArea();

   double CalculatePerimeter();

}



// Class implementing an interface

public class Rectangle : IShape

{

   private double length;

   private double width;
```

```
    public Rectangle(double length, double width)
    {
        this.length = length;
        this.width = width;
    }

    public double CalculateArea()
    {
        return length * width;
    }

    public double CalculatePerimeter()
    {
        return 2 * (length + width);
    }
}

// Class that does not implement the interface
public class Circle
{
    private double radius;

    public Circle(double radius)
```

```
    {
        this.radius = radius;
    }

    public double CalculateArea()
    {
        return Math.PI * radius * radius;
    }

    public double CalculateCircumference()
    {
        return 2 * Math.PI * radius;
    }
}
```

16. What are the advantage of using string Builder class over string class? Give example.

- The StringBuilder class in Java provides several advantages over the String class when it comes to manipulating and modifying strings. Here are some advantages of using StringBuilder:

1. Mutable: StringBuilder objects are mutable, meaning you can modify them by appending, inserting, replacing, or deleting characters without creating a new object. This can be more efficient than creating new String objects, especially when dealing with large or frequently changing strings.
2. Performance: StringBuilder is designed for better performance in scenarios where string manipulation is involved. Since it allows in-place modifications, it avoids the overhead of creating multiple string objects during concatenation or modification operations.
3. Efficient memory allocation: StringBuilder provides a more efficient memory allocation strategy compared to String. When you append characters to a StringBuilder, it automatically adjusts its capacity to accommodate the appended characters, which helps reduce unnecessary memory reallocation.
4. Convenient methods: StringBuilder offers various methods for modifying strings, such as append(), insert(), delete(), replace(), and more. These methods provide flexibility and ease of use when constructing or modifying strings.

Here's an example that demonstrates the advantage of using StringBuilder over String when concatenating multiple strings in a loop

```
// Using StringBuilder

StringBuilder sb = new StringBuilder();

for (int i = 0; i < 10000; i++) {

   sb.append("value").append(i).append(", ");

}

String result = sb.toString();



System.out.println(result);



// Using String concatenation

String str = "";

for (int i = 0; i < 10000; i++) {

   str += "value" + i + ", ";

}



System.out.println(str);
```

In this example, the StringBuilder approach is more efficient because it avoids the repeated creation of intermediate String objects during concatenation. It performs better in terms of both speed and memory usage.

17. What is inheritance? Explain different type of inheritance with examples.

- Inheritance is a fundamental concept in object-oriented programming (OOP) that allows one class to inherit the properties and methods of another class. The class that inherits is called the "subclass" or "derived class," and the class from which it inherits is called the "superclass" or "base class." Inheritance promotes code reusability and allows for the creation of hierarchical relationships between classes.

There are several types of inheritance that can be implemented in OOP:

1. Single Inheritance: Single inheritance is the simplest form of inheritance, where a subclass inherits from a single superclass. The subclass inherits all the public and protected properties and methods of the superclass. Here's an example in Python:

```
class Vehicle:

    def __init__(self, brand):

        self.brand = brand


    def drive(self):

        print("Driving...")


class Car(Vehicle):

    def __init__(self, brand, model):

        super().__init__(brand)

        self.model = model


    def park(self):

        print("Parking...")


my_car = Car("Toyota", "Camry")

print(my_car.brand)  # Output: Toyota

my_car.drive()      # Output: Driving...

my_car.park()       # Output: Parking...
```

In this example, the **Car** class inherits from the **Vehicle** class. The **Car** class inherits the **brand** property and the **drive()** method from the **Vehicle** class.

2. Multiple Inheritance: Multiple inheritance allows a subclass to inherit from multiple superclasses. The subclass inherits properties and methods from all the superclasses. However, multiple inheritance can lead to complexities and conflicts if not used carefully. Here's an example in Python:

```
class Animal:
    def eat(self):
        print("Eating...")

class Flyable:
    def fly(self):
        print("Flying...")

class Bird(Animal, Flyable):
    def __init__(self, name):
        self.name = name

    def chirp(self):
        print("Chirping...")

my_bird = Bird("Sparrow")
my_bird.eat()      # Output: Eating...
my_bird.fly()      # Output: Flying...
my_bird.chirp()    # Output: Chirping...
```

In this example, the **Bird** class inherits from both the **Animal** and **Flyable** classes. The **Bird** class inherits the **eat()** method from the **Animal** class and the **fly()** method from the **Flyable** class.

3. Multilevel Inheritance: Multilevel inheritance involves creating a chain of inheritance, where a subclass becomes the superclass for another subclass. Each subclass inherits the properties and methods of its immediate superclass. Here's an example in Python:

```
class Animal:
    def eat(self):
        print("Eating...")


class Mammal(Animal):
    def feed_milk(self):
        print("Feeding milk...")


class Dog(Mammal):
    def bark(self):
        print("Barking...")


my_dog = Dog()
my_dog.eat()       # Output: Eating...
my_dog.feed_milk()  # Output: Feeding milk...
my_dog.bark()      # Output: Barking...
```

18. Write a program to show the execution sequence of constructor and destructor in multiple inheritance (Include output). Use at least three level of derivation.

- Certainly! Here's an example program that demonstrates the execution sequence of constructors and destructors in multiple inheritance with three levels of derivation:

```
#include <iostream>

using namespace std;


class Base {

public:

   Base() {

      cout << "Base Constructor" << endl;

   }


   ~Base() {

      cout << "Base Destructor" << endl;

   }

};


class Derived1 : virtual public Base {

public:

   Derived1() {

      cout << "Derived1 Constructor" << endl;

   }


   ~Derived1() {

      cout << "Derived1 Destructor" << endl;

   }

};
```

```
class Derived2 : virtual public Base {
public:
    Derived2() {
        cout << "Derived2 Constructor" << endl;
    }

    ~Derived2() {
        cout << "Derived2 Destructor" << endl;
    }
};

class FinalDerived : public Derived1, public Derived2 {
public:
    FinalDerived() {
        cout << "FinalDerived Constructor" << endl;
    }

    ~FinalDerived() {
        cout << "FinalDerived Destructor" << endl;
    }
};

int main() {
```

```
    FinalDerived obj;
    return 0;
}
```

Output:

```
Base Constructor
Derived1 Constructor
Derived2 Constructor
FinalDerived Constructor
FinalDerived Destructor
Derived2 Destructor
Derived1 Destructor
Base Destructor
```

19. What is LINQ? How LINQ is used? Explain with example

LINQ stands for Language-Integrated Query, and it is a feature in the .NET framework that provides a convenient and powerful way to query and manipulate data from different data sources, such as databases, collections, XML, and more. It allows developers to write queries using a uniform syntax regardless of the data source, making it easier to work with and manipulate data in a declarative and expressive manner.

LINQ offers a set of query operators that can be used to perform various operations on data, including filtering, sorting, grouping, joining, and aggregating. These query operators are provided as extension methods on different types, such as IEnumerable<T>, IQueryable<T>, and others, allowing you to chain them together to compose complex queries.

Here's an example to illustrate how LINQ can be used:

Suppose you have a collection of Person objects, and you want to retrieve the names of all adults (people above 18 years of age) from that collection. Without LINQ, you would typically write a loop and manually filter the data. However, with LINQ, you can accomplish the same task in a more concise and readable way.

```
using System;
```

```
using System.Collections.Generic;
using System.Linq;

public class Person
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public class Program
{
    public static void Main()
    {
        List<Person> people = new List<Person>()
        {
            new Person { Name = "John", Age = 25 },
            new Person { Name = "Alice", Age = 18 },
            new Person { Name = "Bob", Age = 30 },
            new Person { Name = "Eve", Age = 15 }
        };

        var adults = from person in people
                     where person.Age > 18
                     select person.Name;
```

```
        foreach (var name in adults)
        {
            Console.WriteLine(name);
        }
    }
}
```

20. Write C# program that reads name age and height of a person and store that info in database named db_2020 batch.

- To create a C# program that reads the name, age, and height of a person and stores that information in a database named "db_2020 batch," you'll need to use a combination of C# programming and a database library like Entity Framework or ADO.NET. In this example, we'll use Entity Framework Core to interact with the database.

Make sure you have the necessary NuGet packages installed:

1. Entity Framework Core: **Microsoft.EntityFrameworkCore** (version 5.0 or later)
2. Database provider package for your chosen database: For example, **Microsoft.EntityFrameworkCore.SqlServer** for SQL Server.

Once you have the packages installed, you can use the following code:

```
using System;

using Microsoft.EntityFrameworkCore;


namespace PersonDatabase

{

  // Define a Person model

  public class Person
```

```
{
    public int Id { get; set; }
    public string Name { get; set; }
    public int Age { get; set; }
    public double Height { get; set; }
}

// Define a DbContext class for your database
public class PersonDbContext : DbContext
{
    public DbSet<Person> People { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        // Set the connection string for your database
        optionsBuilder.UseSqlServer("YourConnectionString");
    }
}

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Enter person details:");
```

```
Console.Write("Name: ");
string name = Console.ReadLine();

Console.Write("Age: ");
int age = int.Parse(Console.ReadLine());

Console.Write("Height: ");
double height = double.Parse(Console.ReadLine());

// Create a new Person object with the entered details
var person = new Person
{
    Name = name,
    Age = age,
    Height = height
};

// Save the person object to the database
using (var context = new PersonDbContext())
{
    context.People.Add(person);
    context.SaveChanges();
}
```

```
            Console.WriteLine("Person details saved to the database.");
        }
    }
}
```

21. What are events? How event is handled in C# ? Explain with example.

- In computer programming, an event is a notification or a signal that indicates that something has happened. Events are commonly used to handle user interactions, system notifications, or any other kind of asynchronous occurrence in a program. In C#, events are implemented using the event keyword and can be subscribed to or handled by event handlers.

Here's an example of how events are handled in C#:

```
using System;


// Step 1: Define a class that will generate events
public class EventGenerator
{
    // Step 2: Define an event using the event keyword
    public event EventHandler MyEvent;


    // Step 3: Define a method to trigger the event
    public void TriggerEvent()
    {
        // Step 4: Raise the event
        OnMyEvent();
```

```
    }

    // Step 5: Define a method to invoke the event
    protected virtual void OnMyEvent()
    {
        // Step 6: Check if there are subscribers to the event
        MyEvent?.Invoke(this, EventArgs.Empty);
    }
}

// Step 7: Define a class that will handle the event
public class EventHandlerClass
{
    // Step 8: Define a method that matches the event signature
    public void HandleEvent(object sender, EventArgs e)
    {
        Console.WriteLine("Event handled!");
    }
}

// Step 9: Demonstrate the event handling
public class Program
{
    public static void Main()
```

```
    {
        EventGenerator eventGenerator = new EventGenerator();
        EventHandlerClass eventHandler = new EventHandlerClass();

        // Step 10: Subscribe to the event by adding an event handler
        eventGenerator.MyEvent += eventHandler.HandleEvent;

        // Step 11: Trigger the event
        eventGenerator.TriggerEvent(); // This will print "Event handled!"
    }
}
```

22. What is abstract class? Write program to illustrate abstract class.

- An abstract class in object-oriented programming is a class that cannot be instantiated directly. It serves as a blueprint for other classes and defines common attributes and behaviors that its subclasses must implement. Abstract classes can have abstract methods, which are methods without an implementation. Subclasses of an abstract class must provide an implementation for all abstract methods defined in the abstract class.

Here's an example program in Python to illustrate an abstract class:

```
from abc import ABC, abstractmethod

# Define an abstract class
class Shape(ABC):
    @abstractmethod
    def calculate_area(self):
        pass
```

```
# Define a concrete class that inherits from the abstract class

class Rectangle(Shape):

    def __init__(self, width, height):

        self.width = width

        self.height = height


    def calculate_area(self):

        return self.width * self.height


# Create objects and call methods

rectangle = Rectangle(5, 3)

print(rectangle.calculate_area())
```

23. Define entity framework . Write program to illustrate code first approach

- Entity Framework is an object-relational mapping (ORM) framework provided by Microsoft. It enables developers to work with relational databases using object-oriented programming concepts. With Entity Framework, you can create, retrieve, update, and delete data from a database using strongly typed .NET objects, without having to write SQL queries directly.

Code First approach is one of the development workflows supported by Entity Framework. In this approach, you start by defining your domain model classes and their relationships, and then Entity Framework generates the database schema based on those classes.

Here's an example of a C# program that demonstrates the Code First approach using Entity Framework:

```
// Step 1: Install Entity Framework package

// Install-Package Microsoft.EntityFrameworkCore


using Microsoft.EntityFrameworkCore;
```

```
// Step 2: Define domain model classes

public class Student

{

   public int StudentId { get; set; }

   public string Name { get; set; }

   public int Age { get; set; }

}



// Step 3: Define DbContext class

public class SchoolContext : DbContext

{

   public DbSet<Student> Students { get; set; }



   protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)

   {


optionsBuilder.UseSqlServer("Server=localhost;Database=SchoolDB;Trusted_Conn
ection=True;");

   }

}



// Step 4: Main program

class Program

{
```

```
    static void Main(string[] args)
    {
        // Step 5: Create an instance of the DbContext
        using (var context = new SchoolContext())
        {
            // Step 6: Create a new student object
            var student = new Student
            {
                Name = "John Doe",
                Age = 20
            };

            // Step 7: Add the student to the DbContext
            context.Students.Add(student);

            // Step 8: Save changes to the database
            context.SaveChanges();
        }
    }
}
```

24. What is razor C# ? Create html page that include razor c# syntax to display current time stamp.

- Razor is a syntax used in ASP.NET, a web development framework developed by Microsoft. It allows you to embed C# code directly into HTML markup, making it

easier to build dynamic web pages. Razor syntax is primarily used in Razor views, which are HTML templates with embedded C# code.

To create an HTML page that includes Razor C# syntax to display the current timestamp, follow these steps:

1. Create a new HTML file with a **.cshtml** extension. For example, you can name it **timestamp.cshtml**.
2. Open the **timestamp.cshtml** file in a text editor or an Integrated Development Environment (IDE) that supports Razor syntax.
3. Inside the file, start by writing the HTML structure. For this example, we'll create a simple page that displays the current timestamp.

```
<!DOCTYPE html>

<html>

<head>

  <title>Current Timestamp</title>

</head>

<body>

  <h1>Current Timestamp</h1>

  <p>The current timestamp is: @DateTime.Now</p>

</body>

</html>
```

4. In the code above, the **@DateTime.Now** expression is Razor syntax. It uses the **DateTime.Now** method in C# to retrieve the current date and time and embeds it within the HTML markup using the **@** symbol.
5. Save the **timestamp.cshtml** file.

To view the output, you'll need to host the ASP.NET application. This can be done using IIS (Internet Information Services) or running it locally using Visual Studio or the .NET Core CLI.

When you navigate to the **timestamp.cshtml** page in a browser, you should see a heading that says "Current Timestamp" and a paragraph below it displaying the current date and time.

25. Write program to make use of stack class to create string stack of size 5 and show stack operation.

- Certainly! Here's an example of a program written in Python that utilizes a stack class to create a string stack of size 5 and demonstrates stack operations such as push, pop, and display:

```
class Stack:

    def __init__(self, size):

        self.size = size

        self.stack = []



    def push(self, item):

        if len(self.stack) < self.size:

            self.stack.append(item)

            print(f"Pushed '{item}' onto the stack.")

        else:

            print("Stack is full. Unable to push item.")



    def pop(self):

        if not self.is_empty():

            item = self.stack.pop()

            print(f"Popped '{item}' from the stack.")

            return item

        else:

            print("Stack is empty. Unable to pop item.")

            return None
```

```
    def is_empty(self):
        return len(self.stack) == 0

    def display(self):
        if not self.is_empty():
            print("Stack contents:")
            for item in reversed(self.stack):
                print(item)
        else:
            print("Stack is empty.")


# Create a stack of size 5
stack = Stack(5)

# Perform stack operations
stack.push("Apple")
stack.push("Banana")
stack.push("Cherry")
stack.push("Date")
stack.push("Elderberry")  # Stack is full at this point
stack.push("Fig")  # Unable to push onto full stack

stack.display()
```

```
stack.pop()

stack.pop()

stack.pop()

stack.pop()

stack.pop()

stack.pop()  # Unable to pop from empty stack



stack.display()
```

26. Write C# program that read tittle, author, price and page of a person and store that info in database named "library".

- Certainly! Here's a C# program that reads the title, author, price, and page information from the user and stores it in a database named "library" using SQL Server.

To run this program, make sure you have the necessary NuGet packages installed: **System.Data.SqlClient**.

```
using System;

using System.Data.SqlClient;



namespace LibraryManagement

{

   class Program

   {

      static void Main(string[] args)

      {
```

```
        Console.WriteLine("Enter the book details:");

        Console.Write("Title: ");
        string title = Console.ReadLine();

        Console.Write("Author: ");
        string author = Console.ReadLine();

        Console.Write("Price: ");
        decimal price = decimal.Parse(Console.ReadLine());

        Console.Write("Page: ");
        int page = int.Parse(Console.ReadLine());

        // Connection string to the SQL Server database
        string connectionString = "Data Source=(local);Initial
Catalog=library;Integrated Security=True";

        using (SqlConnection connection = new SqlConnection(connectionString))
        {
           try
           {
              connection.Open();

              // SQL query to insert book details into the "books" table
```

```
            string query = "INSERT INTO books (Title, Author, Price, Page) VALUES
(@Title, @Author, @Price, @Page)";


            using (SqlCommand command = new SqlCommand(query, connection))
            {
                command.Parameters.AddWithValue("@Title", title);
                command.Parameters.AddWithValue("@Author", author);
                command.Parameters.AddWithValue("@Price", price);
                command.Parameters.AddWithValue("@Page", page);


                int rowsAffected = command.ExecuteNonQuery();


                if (rowsAffected > 0)
                    Console.WriteLine("Book details inserted successfully!");
                else
                    Console.WriteLine("Failed to insert book details.");
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine("Error: " + ex.Message);
        }
    }


    Console.ReadLine();
```

```
    }
  }
}
```

27. Write a features of OOP

- Object-Oriented Programming (OOP) is a programming paradigm that organizes code into objects, which are instances of classes. OOP promotes the concept of encapsulation, inheritance, polymorphism, and abstraction. Here are some key features of OOP:

1. **Classes and Objects**: OOP revolves around the concept of classes and objects. A class is a blueprint or template that defines the properties and behaviors of objects. An object is an instance of a class, representing a specific entity or concept.
2. **Encapsulation**: Encapsulation refers to the bundling of data and methods within a class, where the data (attributes or properties) is hidden from external access and can only be accessed through defined methods. This concept ensures data security and maintains the integrity of the object.
3. **Inheritance**: Inheritance is a mechanism that allows a class to inherit the properties and behaviors of another class, known as the superclass or base class. The class inheriting from the superclass is called the subclass or derived class. Inheritance promotes code reusability and hierarchical organization of classes.
4. **Polymorphism**: Polymorphism means the ability of an object to take on multiple forms or have multiple behaviors. It allows objects of different classes to be treated as objects of a common superclass. Polymorphism enables the use of a single interface to represent different implementations.
5. **Abstraction**: Abstraction focuses on representing the essential features of an object while hiding the unnecessary details. It allows developers to create abstract classes or interfaces that define a set of methods without specifying their implementation. Abstraction helps in managing complexity and facilitates code maintenance.
6. **Modularity**: OOP promotes modularity by breaking down complex problems into smaller, manageable units called objects. Each object encapsulates a specific functionality and can be developed, tested, and maintained independently. This modular approach enhances code readability, reusability, and maintainability.
7. **Message Passing**: Objects communicate with each other by sending messages. A message is a request for an object to perform a specific operation. The receiving object, based on its class implementation, responds to the message by executing the appropriate method. Message passing enables interaction between objects and facilitates the exchange of information.
8. **Polymorphic Relationships**: OOP allows relationships to exist between objects, such as association, aggregation, and composition. These relationships define how objects interact and collaborate with each other. Polymorphic relationships enable flexibility and adaptability in designing systems.
9. **Overloading and Overriding**: OOP supports method overloading and overriding. Overloading allows the creation of multiple methods with the same name but different

parameters within a class. Overriding, on the other hand, allows a subclass to provide a different implementation of a method defined in its superclass. These features enhance code flexibility and customization.

10. **Data Hiding and Access Control**: OOP provides mechanisms for data hiding and access control. By using access modifiers like public, private, and protected, developers can restrict access to certain members (attributes or methods) of a class. This ensures data integrity and allows for better control over the visibility of class components.

These features of OOP collectively contribute to building modular, reusable, and maintainable software systems, making it a widely adopted paradigm in the field of software development.

28. Difference between POP and OOP

Procedure Oriented Programming	Object Oriented Programming
In POP, program is divided into small parts called **functions**.	In OOP, program is divided into parts called **objects**.
In POP, Importance is not given to **data** but to functions as well as **sequence** of actions to be done.	In OOP, Importance is given to the data rather than procedures or functions because it works as a **real world**.
POP follows **Top Down approach**.	OOP follows **Bottom Up approach**.
POP does not have any access specifier.	OOP has access specifiers named Public, Private, Protected, etc.
In POP, Data can move freely from function to function in the system.	In OOP, objects can move and communicate with each other through member functions.
To add new data and function in POP is not so easy.	OOP provides an easy way to add new data and function.
In POP, Most function uses Global data for sharing that can be accessed freely from function to function in the system.	In OOP, data cannot move easily from function to function, it can be kept public or private so we can control the access of data.
POP does not have any proper way for hiding data so it is **less secure**.	OOP provides Data Hiding so provides **more security**.
In POP, Overloading is not possible.	In OOP, overloading is possible in the form of Function Overloading and Operator Overloading.
Example of POP are : C, VB, FORTRAN, Pascal.	Example of OOP are : C++, JAVA, VB.NET, C#.NET.

13. Difference between method **overloading** and **overriding**

Method overloading	Method overridi
1. More than one method with same name, different prototype in same scope is called method overloading.	1. More than one met same name, same p different scope is ca method overriding.
2. In case of method overloading, parameter must be different.	2. In case of method overriding, parame same.
3. Method overloading is the example of compile time polymorphism.	3. Method overriding example of run tim polymorphism.
4. Method overloading is performed within class.	4. Method overriding two classes.
5. In case of method overloading, Return type can be same or different.	5. In case of method o Return type must b
6. Static methods can be overloaded which means a class can have more than one	6. Static methods can overridden, even if a same static metho

Overriding

```
class Dog{
    public void bark(){
        System.out.println("woof ");
    }
}
class Hound extends Dog{
    public void sniff(){
        System.out.println("sniff ");
    }

    public void bark(){
```

Same Method Name, Same parameter

Overloadin

```
class Dog{
    public void bark(){
        System.out.println
    }

    //overloading method
    public void bark(int nu
        for(int i=0; i<num;
            System.out.
    }
}
```

29. Describe the entity in c#

In C#, an entity typically refers to a class or a data structure that represents a concept or an object in a software application. Entities are often used in object-oriented programming to model real-world entities or abstract concepts.

In the context of databases and entity-relationship modeling, an entity represents a table in a database, and each instance of that entity represents a row or record in that table.

When designing an entity in C#, you typically define a class that encapsulates the properties and behaviors associated with the entity. The properties represent the attributes or data associated with the entity, while the methods define the operations or actions that can be performed on the entity.

Here's an example of an entity class in C#:

```
public class Customer
{
    // Properties
    public int Id { get; set; }
    public string Name { get; set; }
    public string Email { get; set; }

    // Methods
    public void PlaceOrder(Order order)
    {
        // Logic to place an order for the customer
    }

    public void UpdateProfile(string newName, string newEmail)
    {
        // Logic to update customer's profile
    }
```

}

29. EXPLAIN THE WORKING OF DATA ACCESS MODELS IN ADO.NET.

In ADO.NET, data access models provide a way to interact with data sources such as databases. There are three primary data access models in ADO.NET: Connected Model, Disconnected Model, and Entity Framework.

ADO.NET provides three primary data access models:

1. Connected Model: In this model, a direct connection is established between the application and the data source. It involves using Connection, Command, and DataReader objects to execute queries and retrieve data from the data source in real-time.
2. Disconnected Model: This model involves fetching data from the data source into a local dataset using a DataAdapter. The data can then be manipulated and analyzed offline within the dataset, and changes can be sent back to the data source when required.
3. Entity Framework: It is an ORM framework that provides a higher level of abstraction for data access. It allows developers to work with conceptual entities and relationships rather than dealing with low-level database interactions directly. It includes an Entity Data Model, ObjectContext/DbContext, and LINQ to Entities for querying and manipulating data.

These models offer different approaches to data access, catering to various requirements and scenarios.