## Introduction Of Shopping Cart

A shopping cart, also known as a shopping trolley or basket, is a software component or system that allows users to collect and manage items while shopping online. The purpose of a shopping cart is to enable customers to add products to their cart, review the selected items, and proceed to the checkout process to complete their purchase. :

1. Product Catalog: The shopping cart interacts with a product catalog or inventory system that contains information about the available products, such as name, description, price, and stock availability.

2. Cart Management: When a user visits our website, they are presented with product listings. They can choose to add desired items to their cart by clicking an "Add to Cart" button associated with each product. The shopping cart system then manages the storage and organization of the selected items.

3. Cart Contents: The shopping cart maintains a record of the added items, typically in the form of a list. Each item in the cart includes details like the product name, quantity, price, and subtotal.

4. Quantity and Price Adjustments: Users can modify the quantity of items in their cart or remove items altogether. The cart updates the quantities and recalculates the subtotal and the total price accordingly.

5. Checkout Process: Once the user has finished adding items to the cart, they can proceed to the checkout process. This involves entering shipping information, selecting a payment method, and reviewing the order details before confirming the purchase.

6. Order Confirmation: After the user confirms the purchase, the shopping cart system generates an order confirmation. It typically includes an order number, a summary of the purchased items, the total price, and any relevant shipping or billing information.

7. Session Persistence: To maintain the user's cart across multiple browsing sessions, the shopping cart system often employs techniques such as browser cookies or user accounts. This allows users to continue shopping or return to their cart at a later time without losing their selected items.

In addition to these core functionalities, shopping cart systems may also include features like discounts or coupon code application, shipping cost calculations, tax calculations, customer registration, and order history tracking.

The design and implementation of a shopping cart system can vary depending on the specific requirements and technologies used. It often involves server-side programming, database management, and integration with payment gateways for secure online transactions.

## Microservices used in Project:-

1. User Management Service: This microservice handles user authentication, registration, and profile management. It may provide endpoints for user registration, login, password reset, and user profile management.
2. Product Catalog Service: This microservice manages the product catalog, including creating, updating, and retrieving product information. It can expose APIs for listing products, retrieving product details, and managing inventory.

3. Order Management Service: The order management microservice is responsible for handling the ordering process, including creating new orders, tracking order status, and managing order fulfillment. It may expose APIs for placing orders, retrieving order details, and updating order status.
4. Payment Service: This microservice integrates with payment gateways and handles payment processing. It may provide APIs for initiating payments, handling payment callbacks, and managing payment transaction history.
5. Notification Service: The notification microservice handles sending notifications to users through various channels such as email, SMS, or push notifications. It may offer APIs for sending notifications, managing notification templates, and tracking delivery status.

## Technological Stack :

1. Development Frameworks :

   For backend - .NET core version 6

   For frontend- node.js

2. Programming Language used – C sharp
3. DBMS used – SQL server Management System (SSMS)
4. For authentication - ASP.NET CORE IDENTITY
5. Web Technology for frontend – Angular
6. For testing – Swagger

## Questions for Evaluation-

Q: What are directives in Angular?
A: They are used to create reusable components or add specific behavior to elements. There are three types of directives in Angular: Component Directives, Attribute Directives, and Structural Directives.

Q: What are pipes in Angular?
A: Pipes in Angular are used for data transformation and formatting. They allow you to transform data before displaying it in the template. Angular provides several built-in pipes for common tasks such as date formatting, number formatting, and string manipulation. You can also create custom pipes to suit your specific needs.

Q: What is a stored procedure?
A: A stored procedure is a set of precompiled SQL statements that are stored in a database. It is a named collection of SQL queries, control statements, and procedural code that can be executed on demand.

Q: What is a component in Angular?

A: In Angular, a component is a building block of an application that controls a part of the user interface. Components are responsible for rendering a view, handling user interactions, and providing data and behavior to other components.

Q: What is a partial class?
A: A partial class is a feature in some programming languages (such as C#) that allows you to split the definition of a class across multiple files. This can be useful when working with large classes or collaborating with other developers on different parts of a class.

Q: What are annotations?
A: In the context of Angular, annotations refer to decorators. Decorators are a way to add metadata to classes, methods, or properties in TypeScript. They are denoted by the @ symbol and are used to enhance the behavior of the target element.

Q: How do you test a component in Angular?
A: In Angular, you can test components using the Angular Testing Library or the TestBed framework. The Angular Testing Library provides utility functions for interacting with components and asserting their behavior in a test environment. The TestBed framework allows you to create a testing module and configure and instantiate components for testing.

Q: Why did you use the code-first approach?
A: The code-first approach is a software development methodology where the focus is on writing code to define the structure and behavior of the application before creating the underlying infrastructure.

Q: What is CORS?
A: CORS (Cross-Origin Resource Sharing) is a security mechanism implemented in web browsers that controls access to resources (such as APIs) on a different domain or origin. It ensures that web applications only make requests to resources on the same domain by default.

Q. what is addscoped?
A. AddScoped is a method in the ASP.NET Core framework used for registering services with scoped lifetime in the dependency injection container. When a service is registered with AddScoped, a new instance of the service is created for each HTTP request within the scope. This means that the same instance will be used throughout the entire request, but different requests will have different instances.

Q. what is the use of swagger?
A. Swagger is an open-source framework that simplifies the documentation, design, and testing of APIs. It provides a user-friendly interface that allows developers to explore and interact with the API without the need for external tools or documentation.

Q. what is a web api?
A. A Web API (Web Application Programming Interface) is a framework for building HTTP-based services that can be accessed by clients, such as web browsers or mobile applications. It allows communication

and data exchange between different software systems over the internet using standard HTTP methods like GET, POST, PUT, and DELETE.

Q. what is REST service?
A. REST (Representational State Transfer) is an architectural style for designing networked applications. A REST service, or RESTful service, is a web service that follows the principles of REST. It relies on the use of standard HTTP methods (GET, POST, PUT, DELETE) to perform operations on resources identified by URLs

Q. what is doctype in HTML?
A. <!DOCTYPE> is an HTML declaration that specifies the document type and version of HTML being used in a web page.

Q. What is different type of CSS style?
A. 1)Inline Styles: These styles are defined directly within the HTML element using the style attribute.
2)Internal Stylesheets: These styles are defined within the <style> tags in the <head> section of an HTML document.
3)External Stylesheets: These styles are defined in separate CSS files and linked to the HTML document using the <link> tag.

Q. what is difference b/w delete, truncate and drop?
A. 1- Delete: The DELETE statement is used to remove specific rows from a table based on a condition. It deletes individual rows while keeping the table structure intact.
2- Truncate: The TRUNCATE statement is used to remove all the rows from a table, effectively emptying the table. It also resets any auto-incrementing IDs or sequences but keeps the table structure.
3- Drop: The DROP statement is used to remove an entire table from the database. It deletes the table along with all its data and associated indexes, triggers, and constraints.

Q. How many types of variables are there in JAVAscript?
A. 1- var variables: These are variables declared using the var keyword. They have function scope or global scope, depending on where they are declared. They can be redeclared and reassigned throughout the scope in which they are defined.
2- let variables: Introduced in ECMAScript 6 (ES6), let variables have block scope. They are limited to the block (within curly braces) where they are defined. They can be reassigned but cannot be redeclared within the same scope.
3- const variables: const variables are also introduced in ES6 and have block scope. They are similar to let variables but cannot be reassigned once they are initialized. However, const variables can still be mutated if they are referencing mutable objects.

Q. what is polymhrphism and inheritance?

A.Poly- It allows methods in different classes to have the same name but perform different actions based on the object type at runtime.
Inheritance- class inherits properties and behaviors from a parent class (superclass).

Q. what is class and structure?
A. a class is a blueprint or template for creating objects. It defines the properties (attributes) and behaviors (methods) that objects of that class will have.
Structure is a value type and a collection of variables of different data types under a single unit.
- by default, all members are public in a structure,
while in a class, they are private by default.
- Structures are value types, whereas classes are reference types in C#.

Q. What is grant and revoke?
A. Grant and revoke are SQL statements used to manage permissions and privileges in a database. Grant is used to give specific privileges or permissions to a user or role, while revoke is used to remove previously granted privileges or permissions from a user or role.

Q. What is doc (doctype) in HTML?
A. It is an "information" to the browser about what document type to expect.

Q. What is state and props?
A. ITS IN REACT NOT IN ANGULAR

Q. What is the difference between ADO.NET and LINQ?
A. -ADO.NET is a data access technology in .NET used to interact with relational databases. It provides classes and APIs for connecting to databases, executing queries, and retrieving or updating data.

-LINQ (Language Integrated Query) is a language feature in .NET that provides a consistent query syntax to query and manipulate data from various data sources. It allows querying data using a SQL-like syntax directly within programming languages like C# or VB.NET.

Q. What is routing?
A. Routing is the process of defining and managing the navigation paths in a web application. It determines how URLs are mapped to specific actions or components in the application

Q.How to do attribute routing?

A. Attribute routing in ASP.NET Core MVC is done by decorating action methods or controllers with attributes like [Route] or [HttpGet]. These attributes allow specifying the URL pattern directly on the action method or controller, providing more flexibility and control over the routing configuration.

Q. What are HTML tag helpers?

A. HTML tag helpers are a feature in ASP.NET Core MVC that simplifies the creation and rendering of HTML elements in Razor views. They allow writing HTML markup with embedded server-side code that can generate dynamic attributes or content based on the application's state. Tag helpers provide a more intuitive and readable way to work with HTML elements compared to traditional HTML helpers or manual HTML generation.

Q. What is an abstract class?
A. It is a class that cannot be instantiated directly but serves as a base or blueprint for derived classes. It can contain abstract methods (methods without implementation) and concrete methods (methods with implementation).

Q. What is the difference between an abstract class and an interface?
A. -An abstract class can have both abstract and non-abstract methods, while an interface can only have method signatures.
-An abstract class can have fields and provide some implementation, while an interface cannot have fields or provide any implementation.
-An abstract class is used when you want to define a common base for derived classes, while an interface is used to define a contract that classes must adhere to.

Q. What is a database?
A. A database is a structured collection of data that is organized and stored in a way that allows efficient retrieval, modification, and management of data.

Q. What is a primary key and a foreign key?
A. - a primary key is a column or a combination of columns that uniquely identifies each record in a table.
- A foreign key, on the other hand, is a column or a combination of columns that establishes a link or a relationship between two tables. It refers to the primary key of another table, creating a relational association between the tables.

Q. What are DML, DDL, and DQL statements?
A. - DML (Data Manipulation Language) statements are used to manipulate or modify data stored in a database. Examples include SELECT, INSERT, UPDATE, and DELETE statements.
- DDL (Data Definition Language) statements are used to define or modify the structure of database objects, such as tables, indexes, and constraints. Examples include CREATE, ALTER, and DROP statements.
- DQL (Data Query Language) statements are used to retrieve data from the database. The most common DQL statement is the SELECT statement.

Q. What is normalization?
A. Normalization is the process of organizing data in a database to minimize redundancy and dependency issues. It involves breaking down a large table into smaller, more manageable tables and defining relationships between them using primary and foreign keys.

Q. What is overriding?
A. Same method name and Same Parameters. The overridden method in the subclass has the same name, return type, and parameters as the method in the superclass. The overridden method is called instead of the superclass's method when the method is invoked on an object of the subclass.

Q. What is overloading?
A. Same method name and Different parameters.

Q. What are HTTP GET, POST, PUT, and DELETE?
A. HTTP Verbs
GET: Retrieves data or a resource from a server. It is used for reading or fetching information.
POST: Sends data to a server to create a new resource or perform an action that may have side effects.
PUT: Sends data to a server to update or replace an existing resource with the provided data.
DELETE: Requests the server to delete a specified resource.

Q. What is Entity Framework?
A. Entity Framework (EF) is an Object-Relational Mapping (ORM) framework provided by Microsoft. It is used to simplify database access and interaction in .NET applications. EF enables developers to work with databases using object-oriented techniques, where database tables are represented as classes and database records are represented as objects.

Q. What is middleware and the sequence of middleware?
A. Middleware in ASP.NET Core is software components that are placed in the request/response pipeline. They handle incoming HTTP requests and responses and can perform various tasks such as authentication, logging, error handling, and more.

The basic sequence of middleware in an ASP.NET Core application is typically as follows:
-Request enters the application: The incoming HTTP request enters the application and starts traversing the middleware pipeline.
-HTTP Server Middleware: This middleware component is responsible for receiving and processing the incoming HTTP request. It handles low-level tasks such as connection management, parsing request headers, and setting up the HttpContext.
-Logging Middleware: This middleware captures information about the incoming request and logs it for debugging, diagnostics, or auditing purposes. It can record details such as request method, URL, headers, and other relevant information.
-Routing Middleware: The routing middleware examines the incoming request URL and determines which endpoint or controller should handle the request based on the defined routes and their associated actions.
-Authentication Middleware: This middleware performs authentication and identity verification for the request. It validates credentials, checks for authentication cookies, or performs other authentication mechanisms to identify the user or client making the request.

-Authorization Middleware: The authorization middleware checks if the authenticated user or client has the necessary permissions or roles to access the requested resource or perform the specified action. It ensures that the user is authorized to proceed.
-Static Files Middleware: This middleware serves static files such as HTML, CSS, JavaScript, images, or other static assets directly from the file system. It locates and returns the requested file if it exists, allowing the application to serve static content efficiently.
-Endpoint Middleware: The endpoint middleware is responsible for executing the endpoint or controller action associated with the incoming request. It calls the appropriate action method and passes control to the corresponding code.
-Application Middleware: This is the custom application-specific middleware that you define to handle various tasks or business logic specific to your application. It can include custom logging, exception handling, data transformation, or any other custom processing.
-Response Middleware: Once the controller action completes its execution, the response middleware takes over. It processes the response generated by the action, sets appropriate headers, formats the response content, and prepares it for delivery back to the client.
-HTTP Server Middleware (Outgoing): Finally, the response is sent back to the HTTP server middleware, which handles the transmission of the response to the client. It manages the connection, writes the response data, and ensures the response is delivered successfully.

Q. What is app.Map()?
A. In ASP.NET Core, app.Map() is a method used to map a specific request path or route to a middleware pipeline. It allows you to conditionally execute a set of middleware based on the matched request path or route. app.Map() takes a path string and a callback function that configures the pipeline for the matched path.

Q. What are tasks and threads?
A. - a thread represents an independent sequence of execution within a process. Threads enable concurrent execution of different parts of a program.
-A task is an abstraction provided by the Task Parallel Library (TPL) in .NET. It represents an asynchronous operation or unit of work that can run concurrently with other tasks. Tasks are higher-level constructs that can be scheduled onto available threads automatically by the underlying runtime.

Q. What are tag helpers?
A. Tag helpers are a feature in ASP.NET Core MVC that allow server-side code to participate in generating and rendering HTML elements. They provide a way to create reusable and flexible HTML elements with associated server-side logic.

Q. What is Razor syntax, and can you provide an example?
A. Razor syntax is a markup syntax used in ASP.NET Core Razor views to embed server-side code within HTML. It provides a concise and expressive way to combine server-side code and HTML markup.

Example of Razor syntax:
html code =>

```
<div>
  <h1>Welcome, @Model.Name!</h1>
  @if (Model.IsAdmin)
  {
    <p>You have administrative privileges.</p>
  }
  else
  {
    <p>You have standard user privileges.</p>
  }
</div>
```

Q. What are SQL constraints?
A. SQL constraints are rules defined on columns or tables in a database that enforce data integrity and consistency. They define certain conditions that data must meet to be stored or modified in the database. Common types of constraints include primary key, foreign key, unique, not null, check, and default constraints.

Q. What are joins?
A. Joins are operations performed on databases to combine rows from multiple tables based on related columns between them. Joins allow retrieving data from multiple tables in a single query by specifying how the tables are related. Common types of joins include inner join, left join, right join, and full outer join.

Q. What is the difference between unique, primary, and foreign keys?
A. - Unique Key: A unique key ensures that the values in a column or a group of columns are unique and not duplicated within the table. It allows null values unless explicitly defined as not null.
- Primary Key: A primary key is a unique key that uniquely identifies each row in a table. It is used to enforce entity integrity and provides a way to reference specific rows from other tables (foreign keys). Primary keys do not allow null values.
- Foreign Key: A foreign key establishes a relationship between two tables by referencing the primary key of another table. It ensures referential integrity and maintains the relationship between related records. Foreign keys refer to the primary key of another table and can have null values to represent optional relationships.

Q. What is a view?
A. In the context of databases, a view is a virtual table that is based on the result of a query. It is created by combining data from one or more tables and can be used like a regular table for querying and manipulating data.

Q. What is MVC and its workflow?
A. MVC stands for Model-View-Controller, which is a software architectural pattern commonly used in web development. It separates the application into three interconnected components:
Model: Represents the data and business logic of the application.

View: Handles the presentation layer and displays the user interface.
Controller: Handles user input, interacts with the model, and updates the view. The workflow in MVC typically involves the user interacting with the view, which triggers a request to the controller. The controller processes the request, interacts with the model to retrieve or modify data, and then updates the view accordingly to render the response.

Q. What is dependency injection?
A. Dependency Injection (DI) is used to implement loose coupling and enhance the maintainability and testability of software systems. It allows objects to define their dependencies through interfaces or abstractions, instead of directly creating or managing their dependencies. The dependencies are then injected or provided by an external entity (a DI container or framework) at runtime. DI helps to improve code modularity, flexibility, and the ability to swap dependencies without modifying the dependent code.

Q. What is JWT token and its workflow?
A. JWT (JSON Web Token) is a compact and self-contained security token used for securely transmitting information between parties as a JSON object. It consists of three parts: a header, a payload, and a signature.
The workflow of JWT involves the following steps:

- Authentication: The user provides credentials, which are verified by the server.

- Token Generation: Upon successful authentication, the server generates a JWT token containing relevant claims (user ID, roles, etc.).
- Token Transmission: The server sends the JWT token back to the client, typically as part of the response payload.
- Subsequent Requests: The client includes the JWT token in the Authorization header of subsequent requests.
- Token Verification: The server validates the JWT token's signature, integrity, and expiration, and extracts the necessary claims to authenticate and authorize the request.

Q. What is exception handling?
A. Exception handling is a mechanism used in programming to catch and handle unexpected or exceptional situations that may occur during the execution of code. It allows developers to anticipate and gracefully handle errors, exceptions. Exception handling involves capturing, processing, and potentially recovering from exceptions by using try-catch blocks, finally blocks, and exception handling constructs provided by the programming language.

Q. What is the Startup class?
A. In ASP.NET Core, the Startup class is a crucial component that configures the application's services and middleware pipeline. It is responsible for setting up the DI container, configuring middleware components, registering services, and defining the application's request handling pipeline. The Startup class is typically found in the application's entry point and contains methods such as ConfigureServices() and Configure().

Q. What is routing?
A. it is used to determine which code or component should handle a specific URL or route. Routing maps URL patterns to controllers allowing the application to generate dynamic responses based on the requested resource.

Q. What are the design patterns used in .NET Core?
A. In .NET Core, several design patterns are commonly used, including:
- Dependency Injection (DI) pattern: Used to decouple dependencies and improve testability and maintainability.
- Repository pattern: Provides an abstraction layer between the data access layer and the rest of the application.
- Factory pattern: Used to create objects based on specific conditions or configurations.
Singleton pattern: Ensures that only one instance of a class is created and shared throughout the application.

- Builder pattern: Simplifies the creation of complex objects by separating the construction process from the object's representation.

- Observer pattern: Enables one-to-many communication between objects, where changes in one object trigger updates in dependent objects.
- Decorator pattern: Allows adding new behavior or functionality to an object dynamically at runtime.
- Strategy pattern: Defines a family of interchangeable algorithms and allows selecting the appropriate one at runtime based on specific conditions.

Q. What are indexes and their types?
A. Indexes in databases are data structures that improve the retrieval speed of records from tables. They are created on one or more columns of a table and facilitate quick data lookup by creating a reference to the physical location of the data.
Types of indexes include:

- Clustered Index: Determines the physical order of data rows in a table. A table can have only one clustered index.

- Non-Clustered Index: Creates a separate structure that includes the indexed column(s) and a reference to the data rows.
- Unique Index: Ensures that the indexed column(s) contain unique values, allowing fast retrieval and enforcing uniqueness.
- Composite Index: Uses multiple columns as a key for indexing, providing efficient querying on multiple columns simultaneously.
- Covered Index: Includes all the necessary columns of a query in the index itself, eliminating the need for a further lookup in the table.

Q. What is a repository?

A. The repository pattern is a design pattern USEDto abstract and encapsulate the data access layer. It provides a consistent and structured way to interact with data storage (such as databases) without exposing the underlying implementation details. A repository acts as an intermediary between the application's business logic and the data source, providing methods to perform CRUD (Create, Read, Update, Delete) operations on entities. It helps improve separation of concerns, code organization, and testability.

Q. How to define global variables in Angular?
A. BY using a shared service to store and manage global state or variables. The shared service can be injected into components and accessed to read or modify the global variables. By using a shared service, the variables can be shared across multiple components and remain consistent throughout the application's lifecycle.

Q. What are services in Angular?
A. In Angular, services are classes that provide reusable functionality and data to components or other services. They are used to encapsulate and separate business logic, data access, or other common tasks that are not specific to a single component. Services promote code reusability, modular architecture, and help maintain a separation of concerns. They can be injected into components using dependency injection to access their methods and properties. Services in Angular are typically registered at the application level or module level, ensuring they are available throughout the application.

Q. How many types of access specifiers do we have in C#?
A. In C#, there are five types of access specifiers:
-public: Allows access to the member from any Class.
-private: Limits access to the member within the same class or struct.
-protected: Allows access to the member within the same class and its derived classes.
-internal: Allows access to the member within the same project ( assembly)
-protected internal: Allows access to the member within the same project (assembly) or its derived classes.

Q. What is abstraction?
A. Abstraction is hiding unnecessary details.  Abstraction  allows the creation of abstract classes or interfaces to define common behavior and characteristics.

Q. What is compile time polymorphism?
A. Compile-time polymorphism, also known as method overloading, is a feature in programming languages that allows multiple methods with the same name but different parameter lists to be defined in a class. During compilation, the appropriate method to execute is determined based on the number, order, and types of arguments provided. The decision is made at compile-time, hence the name compile-time polymorphism.

Q. What is static polymorphism and dynamic polymorphism?
A. - Static polymorphism, also known as early binding or method overloading, occurs when the compiler determines the method to be called based on the method signature at compile-time. The decision is made based on the static type of the object or arguments.
- Dynamic polymorphism, also known as late binding or method overriding, occurs when the method to be called is determined at runtime based on the dynamic type of the object. It allows a subclass to provide its own implementation of a method defined in its superclass. The decision is made based on the actual type of the object at runtime.

Q. What is meant by garbage collection?
A. Garbage collection is an automatic memory management process performed by the runtime environment (such as the .NET Common Language Runtime) to reclaim memory occupied by objects that are no longer in use. It involves identifying and freeing up memory occupied by objects that are no longer reachable or referenced by any active part of the program.

Q. What is an interface?
A. It specifies the methods that a class should have without providing any implementation details. An interface allows unrelated classes to implement the same set of methods, enabling polymorphism and providing a way to achieve multiple inheritance in languages that do not support it directly.

Q. Define a superclass.
A. A superclass, also known as a base class or parent class, is a class from which other classes are derived or inherited.

Q. Are there any limitations of inheritance?
A. Yes, inheritance has some limitations:
- Inheritance establishes a tight coupling between the superclass and its subclasses, which can limit flexibility and increase the risk of ripple effects when modifying the superclass.
- Inheritance can lead to code duplication if multiple levels of inheritance are used, as subclasses inherit all the members of their ancestors.
- Inheritance can make the code more complex and harder to understand, especially when deep inheritance hierarchies are involved.
- Inheritance does not allow a class to inherit multiple classes in languages that do not support multiple inheritance directly. However, this limitation can be overcome by using interfaces or abstract classes.

Q. What is a copy constructor?
A. A copy constructor is a special constructor defined in a class that allows creating a new object by copying the values of another object of the same class. The copy constructor typically takes a single parameter of the same class type and performs a member-wise copy of the object's data.

Q. What is a destructor?
A. A destructor is a special method defined in a class that is automatically called when an object is being destroyed or going out of scope. It is used to release resources, such as memory, files, or network

connections, that the object acquired during its lifetime. In some programming languages, like C++, destructors are explicitly defined using the tilde (~) symbol.

Q. What are the various types of inheritance?
A. The various types of inheritance include:
- Single Inheritance: A class inherits from a single superclass.
- Multiple Inheritance: A class inherits from multiple superclasses, which is supported in some programming languages but not in others (such as C# and Java).
- Multilevel Inheritance: A class inherits from a superclass, and that superclass can further inherit from another superclass, forming a hierarchy.

Q. Is it always necessary to create objects from a class?
A. No, it is not always necessary to create objects from a class. Some classes serve as base or utility classes that provide shared functionality or static methods that can be accessed without creating an instance of the class. In such cases, the class can be used directly without instantiating objects.

Q. What is Compile-time Polymorphism and Runtime Polymorphism?
A. - Compile-time polymorphism, also known as static polymorphism, occurs when the compiler selects the appropriate method or function to execute based on the number, order, and types of arguments passed at compile-time. Method overloading is an example of compile-time polymorphism.
- Runtime polymorphism, also known as dynamic polymorphism, occurs when the method or function to be executed is determined at runtime based on the actual type of the object. Method overriding, achieved through inheritance and virtual methods, is an example of runtime polymorphism.

Q. What is encapsulation?
A. Encapsulation is combining data and the methods that operate on that data into a single unit called an object. It encapsulates the data and behavior, allowing access to the data only through predefined methods or properties. Encapsulation provides data abstraction, hiding the internal details of an object and protecting the data from direct manipulation, ensuring data integrity and promoting code modularity.

Q. What are the main features of OOPs?
A. The main features of object-oriented programming (OOP) are:
- Encapsulation: The bundling of data and methods into a single unit (object).
- Inheritance: The ability to create new classes (derived or child classes) from existing classes (base or parent classes).
- Polymorphism: The ability of objects of different types to be treated as objects of a common base type.
- Abstraction: The representation of complex real-world objects or concepts as simplified models.
Modularity: The ability to break down large systems into smaller, manageable units (classes or modules).
Reusability: The ability to reuse existing code or components in different parts of an application or in different applications.

Q. What are Generic Collections?

A. Generic collections is collections that can store and manipulate objects of a specific type. They are part of the System.Collections.Generic namespace and include classes such as List<T>, Dictionary<TKey, TValue>, Queue<T>, and HashSet<T>. Generic collections enable stronger type checking and eliminate the need for explicit type casting when working with collections.

Q. Why do we use Enums?
A. Enums are used to define a set of named constant values that represent a finite set of related items or options. Enums provide a way to give meaningful names to these values, making the code more readable, maintainable, and self-explanatory. They help enforce type safety, improve code clarity, and eliminate the possibility of invalid values by restricting the possible options to a predefined set.

Q. What are Custom Exceptions?
A. Custom exceptions are exceptions that are created by developers to represent specific error conditions or exceptional scenarios in their applications. Custom exceptions are derived from the base Exception class and can be defined with additional properties, methods, or custom error messages. They allow developers to handle application-specific errors more precisely and provide meaningful information to users or logging systems.

Q. Explain virtual class and virtual method.
A. Not in C# . So don’t know about it. Its used in C++ .
In the context of C#, the term "virtual class" is not applicable. However, in some other programming languages like C++, a virtual class is a class that acts as a base class and can have one or more virtual methods. Derived classes can override these virtual methods to provide their own implementation. virtual method is a method declared in a base class with the virtual keyword. It allows derived classes to provide their own implementation of the method by using the override keyword. The decision on which implementation to invoke is determined at runtime based on the actual type of the object, enabling polymorphism.

Q. Why do we use the finally keyword in exception handling?
A. The finally keyword is used in exception handling to define a block of code that will be executed regardless of whether an exception is thrown or not. It ensures that certain cleanup or finalization tasks are performed, such as releasing resources, closing connections, or closing files. The finally block is guaranteed to execute even if an exception occurs within the corresponding try or catch blocks, allowing developers to handle exceptional cases and ensure necessary cleanup operations are always performed.

Q. What are delegates?
A. Delegates are reference types that hold references to methods. They allow methods to be treated as first-class objects, which can be passed as parameters, stored in variables, and invoked dynamically. Delegates provide a way to achieve callback mechanisms, event handling, and functional programming concepts like function composition and higher-order functions.

Q. What is method hiding?

A. Method hiding occurs when a derived class defines a method with the same name as a method in its base class, effectively hiding the base class method. Method hiding is achieved by using the new keyword when declaring the method in the derived class. The method in the derived class does not override the base class method; instead, it provides a new implementation specific to the derived class. Method hiding can lead to confusion and is generally discouraged in favor of method overriding using the override keyword.

Q. Abstract Classes vs Interfaces
A. - Abstract Classes:
An abstract class is a class that cannot be instantiated and serves as a blueprint for derived classes.
It can contain both abstract and non-abstract members.
Derived classes must provide implementations for all abstract members.
It can provide default implementations for common behavior.
A class can inherit from only one abstract class.

- Interfaces:

An interface is  that defines a set of methods that a class must implement.
It can only contain method signatures, properties, events, and indexers but no implementation.
A class can implement multiple interfaces.
It enforces a clear separation of concerns and allows polymorphism.
It does not provide default implementations; each implementing class must define its own implementation.

Q. Explain Reflection.
A. Reflection is that allows examining and manipulating metadata of types (classes, methods, properties, etc.) at runtime. It provides a way to introspect and discover information about types and their members dynamically.  It is commonly used for tasks like runtime type discovery, building generic frameworks, creating dynamic instances, and performing advanced debugging or testing operations.

Q. What is a Dictionary in C#?
A. A Dictionary is a collection class that represents a collection of key-value pairs, where each key is unique within the dictionary. It is part of the System.Collections.Generic namespace.

Q. Explain Queue and Stack.
A. - Queue:
A Queue is a collection class that represents a first-in, first-out (FIFO) data structure.
Elements are added to the end of the queue and removed from the beginning.
It follows the principle of "first come, first served."
Common operations include Enqueue (add an element to the end) and Dequeue (remove an element from the beginning).

- Stack:

A Stack is a collection class that represents a last-in, first-out (LIFO) data structure.
Elements are added to and removed from the top of the stack.

It follows the principle of "last in, first out."
Common operations include Push (add an element to the top) and Pop (remove an element from the top).

Q. List collection class in C#
A. The List<T> collection class in C# is a generic dynamic array that provides a resizable and indexable collection of elements. It is part of the System.Collections.Generic namespace. List<T> allows adding, removing, and manipulating elements in a sequence. List<T> is commonly used when the size of the collection is not known in advance and needs to grow or shrink dynamically.

Q. When do we use the base keyword?
A. The base keyword in C# is used to refer to the base class or parent class within a derived class. It is primarily used in :
- Accessing base class members: The base keyword allows accessing base class methods, properties, and fields from the derived class.
- Invoking base class constructors: The base keyword is used to invoke a specific constructor of the base class when creating an instance of the derived class. It ensures that the initialization logic of the base class is executed before the derived class-specific logic.

Q. Method overriding vs hiding:
A. - Method overriding is WHERE a derived class provides a new implementation of a method that is already defined in its base class. The method to be executed is determined at runtime based on the actual type of the object.
-Method hiding occurs when a derived class defines a method with the same name as a method in its base class, effectively hiding the base class method. Method hiding is achieved by using the new keyword when declaring the method in the derived class. The method in the derived class does not override the base class method; instead, it provides a new implementation specific to the derived class. The decision on which method to invoke is determined by the compile-time type of the reference.

Q. Static & Instance members:
A. - Static members: They are declared using the static keyword and can be accessed directly on the type without creating an instance. Static members include fields, properties, methods, and nested types. They are shared among all instances of the type and can be accessed using the type name.
- Instance members: They do not have the static keyword and can only be accessed through an instance of the class. Instance members include instance fields, properties, methods, and nested types. Each instance of the class has its own set of instance member values.

Q. Why do we use namespaces?
A. Namespaces in C# are used to organize and group related types (classes, interfaces, enums, etc.) into logical units. They provide a way to avoid naming conflicts between types with the same name by providing a unique context for the type names.

Q. Multithreading in C#:
A. Multithreading in C# involves executing multiple threads concurrently within a single program. Threads are lightweight units of execution that run independently, allowing parallel or concurrent execution of code. Multithreading is beneficial for achieving responsiveness, improving performance, and utilizing the capabilities of modern multi-core processors. C# provides various classes and APIs in the System.Threading namespace to create and manage threads, synchronize access to shared resources, and coordinate the execution of multiple threads.

Q. Async and await keywords in C# and why we use them:
A. The async and await keywords in C# are used for asynchronous programming. They simplify the process of writing asynchronous code by providing a way to write asynchronous operations that resemble synchronous code.
- async is used to define a method that can be executed asynchronously. It allows the method to use the await keyword within its body.
- await is used to await the completion of an asynchronous operation without blocking the execution of the method. It indicates a point at which the method can be suspended and resumed later when the awaited operation completes.

- Asynchronous programming with async and await allows the application to be more responsive by not blocking the main thread, enabling concurrent execution of multiple tasks, and efficiently utilizing system resources.

- Asynchronous MEANS it is a non-blocking architecture, which means it doesn't block further execution while one or more operations are in progress.

Q. When do we use Lambda expressions in C#?
A. They are typically used in :
- Functional programming: Lambda expressions enable functional programming techniques such as passing functions as parameters or returning functions from methods.
- LINQ queries: Lambda expressions are commonly used in LINQ (Language Integrated Query) queries to define predicates, projections, or ordering functions.
- Event handlers: Lambda expressions can be used to define event handlers inline without the need for explicitly defining separate methods.
- Asynchronous programming: Lambda expressions can be used with asynchronous methods to define callback functions or actions to be executed when the asynchronous operation completes.