.NET and Related Concepts: A Comprehensive Guide

1 Class

A class serves as a blueprint or template for creating objects in object-oriented programming (OOP). It encapsulates data (fields or properties) and behavior (methods or functions) that define the characteristics and actions of the objects instantiated from it. A class provides a structured way to model real-world entities in software development.

1.1 Key Characteristics

- Defines properties to store data.
- Defines methods to perform operations.
- Acts as a reusable template for creating multiple objects.

1.2 Real-World Example

Consider a class representing a `Car`. The class defines attributes like color and speed, and behaviors like driving or stopping.

```
public class Car
{
    // Properties
    public string Color { get; set; }
    public int Speed { get; set; }

    // Methods
    public void Drive()
    {
        Console.WriteLine($"The {Color} car is driving at {Speed}
           mph.");
    }

    public void Stop()
    {
        Console.WriteLine("The car has stopped.");
    }
}
```

Listing 1: Example of a Car Class

1.3 Usage

To create an object from the `Car` class:

```
Car myCar = new Car();
myCar.Color = "Red";
myCar.Speed = 60;
myCar.Drive(); // Output: The Red car is driving at 60 mph.
```

2 Object

An object is a concrete instance of a class. It represents a specific entity that has the properties and behaviors defined by its class, with actual values assigned to its properties.

2.1 Key Characteristics

- Holds real-time data as defined by the class.
- Can invoke methods defined in the class.
- Each object is independent with its own state.

2.2 Real-World Example

Using the `Car` class, you can create multiple objects, each with different property values.

```
Car car1 = new Car();
car1.Color = "Blue";
car1.Speed = 50;
car1.Drive(); // Output: The Blue car is driving at 50 mph.

Car car2 = new Car();
car2.Color = "Green";
car2.Speed = 70;
car2.Drive(); // Output: The Green car is driving at 70 mph.
```

Listing 2: Creating Car Objects

3 Types of Authentication

Authentication verifies the identity of a user or system. In .NET applications, several authentication mechanisms are commonly used.

3.1 Windows Authentication

- Uses Windows credentials (e.g., Active Directory accounts).
- Ideal for intranet applications within a corporate network.
- Secure and seamless for Windows-based environments.
- Example: Company intranet portals.

3.2 Forms Authentication

- Requires users to provide credentials via a login form (username/password).
- Common in web applications.
- Credentials are validated against a database or identity provider.
- Example: E-commerce websites with user login pages.

3.3 JWT Authentication (Token-Based)

- Uses JSON Web Tokens (JWT) for stateless authentication.
- Common in APIs and microservices.
- Tokens contain claims (user info) and are signed for security.
- Example: RESTful APIs securing endpoints with Bearer tokens.

3.4 OAuth

- Allows third-party authentication (e.g., login with Google, Facebook).
- Delegates authentication to an external provider.
- Securely shares user data without exposing credentials.
- Example: Gmail's "Sign in with Google" feature.

4 Object-Oriented Programming (OOP) Concepts

OOP is a programming paradigm based on objects and classes. The four core principles are:

4.1 Encapsulation

- Combines data and methods into a single unit (class).
- Restricts direct access to data using access modifiers (e.g., `private`, `public`).
- Example: A `BankAccount` class hides the balance and exposes it via methods like `Deposit()` or `GetBalance()`.

```
public class BankAccount
{
    private decimal balance;

    public void Deposit(decimal amount)
    {
        if (amount > 0) balance += amount;
    }

    public decimal GetBalance()
    {
        return balance;
    }
}
```

Listing 3: Encapsulation Example

4.2 Abstraction

- Hides complex implementation details and exposes only necessary features.
- Achieved using abstract classes or interfaces.
- Example: A `Vehicle` abstract class defines a `Drive()` method, but each vehicle type implements it differently.

4.3 Inheritance

- Allows a class to inherit properties and methods from another class.
- Promotes code reuse and hierarchical relationships.
- Example: A `Car` class inherits from a `Vehicle` class.

```
public class Vehicle
{
    public string Brand { get; set; }
    public virtual void Drive()
    {
        Console.WriteLine("Vehicle is moving.");
    }
}

public class Car : Vehicle
{
    public override void Drive()
    {
        Console.WriteLine($"{Brand} car is driving.");
    }
}
```

Listing 4: Inheritance Example

4.4 Polymorphism

- Allows methods to perform differently based on the object calling them.
- Types: Compile-time (method overloading) and runtime (method overriding).
- Example: The `Drive()` method behaves differently for `Car` and `Bike`.

```
Vehicle myCar = new Car { Brand = "Toyota" };
myCar.Drive(); // Output: Toyota car is driving.
```

Listing 5: Polymorphism Example

5 Versions of Currently Used Tools

The following tools are commonly used in .NET development, with their latest versions as of June 2025:

- **.NET**: .NET 6, .NET 7, .NET 8 (latest). Use `dotnet -version` to check.
- **Visual Studio**: Visual Studio 2022, Visual Studio 2025 (preview or stable).
- **SQL Server**: SQL Server 2019, SQL Server 2022.
- **Angular**: Angular 16, Angular 17.
- **Entity Framework Core**: EF Core 6, EF Core 7, EF Core 8.

6 Entity and Its Purpose

An *entity* in Entity Framework (EF) is a class that maps to a database table. It represents a data model and simplifies database operations using object-oriented code.

6.1 Purpose

- Eliminates the need for raw SQL queries for common CRUD operations.
- Provides a strongly-typed model for database interactions.
- Supports LINQ queries for data retrieval.

6.2 Example

A `Student` entity maps to a `Students` table in the database.

```
public class Student
{
    public int Id { get; set; }
    public string Name { get; set; }
    public int Age { get; set; }
}
```

Listing 6: Entity Example

6.3 Usage with EF Core

```
public class SchoolContext : DbContext
{
    public DbSet<Student> Students { get; set; }
}

using (var context = new SchoolContext())
{
    context.Students.Add(new Student { Name = "John", Age = 20 });
    context.SaveChanges();
}
```

Listing 7: Using Entity with EF Core

7 POST vs. GET Methods

HTTP methods `GET` and `POST` serve different purposes in web applications.

7.1 Comparison

- **Purpose**: `GET` retrieves data; `POST` submits data.
- **Parameters**: `GET` sends parameters in the URL; `POST` sends them in the request body.
- **Idempotent**: `GET` is idempotent (repeated calls produce the same result); `POST` is not.
- **Security**: `GET` is less secure (parameters visible); `POST` is more secure.

7.2 Examples

- `GET /api/students/1`: Fetches student with ID 1.
- `POST /api/students`: Creates a new student record.

8 Entity Framework vs. Stored Procedures

Entity Framework (EF) and stored procedures are two approaches to interact with databases.

8.1 Comparison

- **Code Style**: EF uses object-oriented C# code; stored procedures use SQL.

- **Flexibility**: EF is easier to maintain; stored procedures are harder to modify.
- **Performance**: EF is slower for complex queries; stored procedures are optimized for complex logic.
- **Use Case**: EF for simple CRUD; stored procedures for complex joins or batch operations.

8.2 Example: Stored Procedure

```
CREATE PROCEDURE GetStudentsByAge
    @Age INT
AS
BEGIN
    SELECT * FROM Students WHERE Age > @Age;
END
```

Listing 8: Stored Procedure Example

9 .NET Core vs. .NET Framework

.NET Core (now .NET 5/6/7/8) and .NET Framework are two runtime environments for .NET applications.

9.1 Comparison

- **OS Support**: .NET Framework is Windows-only; .NET Core is cross-platform (Windows, Linux, macOS).
- **Performance**: .NET Framework is slower; .NET Core is optimized for performance.
- **Development Focus**: .NET Framework for legacy desktop/enterprise apps; .NET Core for web, microservices, and cloud.

9.2 Recommendation

Use .NET 6 or later for modern, scalable applications.

10 ADO.NET

ADO.NET is a data access technology in .NET for interacting with databases using low-level SQL commands and connections.

10.1 Key Components

- `SqlConnection`: Establishes a connection to the database.
- `SqlCommand`: Executes SQL queries or stored procedures.

- `SqlDataReader`: Reads data from query results.

10.2 Example

```
using (SqlConnection con = new SqlConnection("connection string"))
{
    con.Open();
    SqlCommand cmd = new SqlCommand("SELECT * FROM Students", con);
    SqlDataReader reader = cmd.ExecuteReader();
    while (reader.Read())
    {
        Console.WriteLine(reader["Name"]);
    }
}
```

Listing 9: ADO.NET Example

11 Return Types in .NET

A return type specifies the type of value a method returns. If no value is returned, the method is declared as `void`.

11.1 Examples

```
public int Add(int a, int b) // Returns an integer
{
    return a + b;
}

public void PrintMessage() // Returns nothing
{
    Console.WriteLine("Hello, World!");
}
```

Listing 10: Return Type Examples

12 API Methods

Common HTTP methods used in RESTful APIs:

- `GET`: Retrieves data (e.g., `GET /api/products`).
- `POST`: Creates data (e.g., `POST /api/products`).
- `PUT`: Updates an entire resource (e.g., `PUT /api/products/1`).
- `PATCH`: Updates part of a resource (e.g., `PATCH /api/products/1`).
- `DELETE`: Deletes data (e.g., `DELETE /api/products/1`).

13 Razor Pages

Razor Pages is a page-based framework in ASP.NET Core for building web applications. It combines HTML and C# code in `.cshtml` files.

13.1 Structure

- `Index.cshtml`: Contains HTML and Razor syntax for the UI.
- `Index.cshtml.cs`: Contains the code-behind logic (e.g., handling form submissions).

13.2 Example

```
@page
@model IndexModel
<h1>Hello, @Model.Message</h1>
```

Listing 11: Razor Page Example

```
public class IndexModel : PageModel
{
    public string Message { get; set; }

    public void OnGet()
    {
        Message = "Welcome to Razor Pages!";
    }
}
```

Listing 12: Razor Page Code-Behind

14 ASP.NET Life Cycle

The ASP.NET life cycle describes the stages a web request goes through.

14.1 Web Forms Life Cycle

- **Page Request**: Request arrives at the server.
- **Start**: Initializes page properties.
- **Initialization**: Loads controls and view state.
- **Load**: Loads page and control data.
- **Postback**: Handles user interactions.
- **Rendering**: Generates HTML output.
- **Unload**: Cleans up resources.

14.2 ASP.NET Core

Uses a middleware pipeline:

- Request Middleware Controller Action Response.

15 Namespace

A namespace is a logical container for organizing related classes, interfaces, and other types to prevent naming conflicts.

15.1 Example

```
namespace StudentApp.Models
{
    public class Student
    {
        public int Id { get; set; }
        public string Name { get; set; }
    }
}
```

Listing 13: Namespace Example

15.2 Usage

```
using StudentApp.Models;
Student student = new Student();
```

16 Filters in ASP.NET Core

Filters allow custom logic to execute before or after controller actions.

16.1 Types

- **Authorization Filter**: Validates user permissions.
- **Action Filter**: Modifies action input/output.
- **Exception Filter**: Handles errors.

16.2 Example

```
public class LogActionFilter : IActionFilter
{
    public void OnActionExecuting(ActionExecutingContext context)
    {
        Console.WriteLine("Action executing.");
    }
```

```
7
8       public void OnActionExecuted(ActionExecutedContext context)
9       {
10          Console.WriteLine("Action executed.");
11      }
12  }
```

Listing 14: Action Filter Example

17 LINQ

LINQ (Language Integrated Query) enables SQL-like queries in C# for querying data sources like collections, databases, and XML.

17.1 Example

```
1  var students = new List<Student>
2  {
3      new Student { Name = "John", Age = 20 },
4      new Student { Name = "Jane", Age = 17 }
5  };
6
7  var adultNames = students
8      .Where(s => s.Age > 18)
9      .Select(s => s.Name);
10 // Result: ["John"]
```

Listing 15: LINQ Example