# Summary of Key C# Topics

## Types of Classes

1. **Abstract Class**:
   * Cannot be instantiated.
   * Can include abstract methods (no body) and concrete methods.
   * Example:
   * public abstract class Shape  
     {  
      public abstract double CalculateArea();  
      public void Display() => Console.WriteLine("Shape displayed");  
     }
2. **Sealed Class**:
   * Cannot be inherited.
   * Example:
   * public sealed class Utility  
     {  
      public void Execute() => Console.WriteLine("Executing...");  
     }
3. **Static Class**:
   * Cannot be instantiated.
   * Contains only static members.
   * Example:
   * public static class MathHelper  
     {  
      public static int Add(int a, int b) => a + b;  
     }
4. **Partial Class**:
   * Allows a class to be split across multiple files.
5. **Interfaces**:
   * Defines a contract that implementing classes must fulfill.
   * Example:
   * public interface ILogger  
     {  
      void Log(string message);  
     }  
       
     public class ConsoleLogger : ILogger  
     {  
      public void Log(string message) => Console.WriteLine(message);  
     }

## Generics

* Provides type safety and reusability.
* Example:
* public class GenericRepository<T>  
  {  
   private List<T> \_items = new List<T>();  
   public void Add(T item) => \_items.Add(item);  
   public IEnumerable<T> GetAll() => \_items;  
  }
* **Generic Constraints**:
  + You can restrict the types used in generics.
  + Example:
  + public class GenericService<T> where T : class  
    {  
     public T Instance { get; set; }  
    }

## File System in Depth

* Reading and writing files:
* using System.IO;  
    
  // Writing to a file  
  File.WriteAllText("example.txt", "Hello, World!");  
    
  // Reading from a file  
  string content = File.ReadAllText("example.txt");  
  Console.WriteLine(content);
* Directory operations:
* Directory.CreateDirectory("NewFolder");  
  var files = Directory.GetFiles(".");
* **FileStream for Large Files**:
* using (FileStream fs = new FileStream("largefile.txt", FileMode.OpenOrCreate))  
  {  
   byte[] buffer = Encoding.UTF8.GetBytes("Large content");  
   fs.Write(buffer, 0, buffer.Length);  
  }

## Data Serialization

1. **JSON Serialization**:

* using System.Text.Json;  
    
  var obj = new { Name = "John", Age = 30 };  
  string json = JsonSerializer.Serialize(obj);  
  var deserialized = JsonSerializer.Deserialize<dynamic>(json);

1. **XML Serialization**:

* using System.Xml.Serialization;  
  using System.IO;  
    
  var serializer = new XmlSerializer(typeof(Person));  
  using (var writer = new StreamWriter("person.xml"))  
   serializer.Serialize(writer, new Person { Name = "John", Age = 30 });

1. **Binary Serialization**:

* using System.Runtime.Serialization.Formatters.Binary;  
    
  var formatter = new BinaryFormatter();  
  using (var stream = new FileStream("data.bin", FileMode.Create))  
   formatter.Serialize(stream, new Person { Name = "John", Age = 30 });

## Base Library Features

* **System.Collections.Generic** for collections.
* **System.IO** for file handling.
* **System.Linq** for LINQ.
* **System.Text** for string manipulations.
* **System.Threading** for multithreading.

## Lambda Expressions

* Inline, anonymous functions.
* Example:
* Func<int, int> square = x => x \* x;  
  Console.WriteLine(square(5));
* Common usage:
* var numbers = new List<int> { 1, 2, 3, 4 };  
  var evens = numbers.Where(n => n % 2 == 0).ToList();

## Extension Methods

* Add functionality to existing types without inheritance.
* Example:
* public static class StringExtensions  
  {  
   public static bool IsNullOrEmpty(this string str) => string.IsNullOrEmpty(str);  
  }
* Usage:
* string text = null;  
  Console.WriteLine(text.IsNullOrEmpty());

## LINQ

1. **With List**:

* var numbers = new List<int> { 1, 2, 3, 4, 5 };  
  var evens = numbers.Where(x => x % 2 == 0);

1. **With DataTable**:

* var table = new DataTable();  
  table.Columns.Add("Name");  
  table.Rows.Add("John");  
  var query = table.AsEnumerable().Where(row => row["Name"].ToString() == "John");

1. **LINQ Queries**:
   * Query syntax:
   * var query = from num in numbers  
      where num > 2  
      select num;

## ORM Tools

* **Entity Framework**:
  + Example:
  + using (var context = new MyDbContext())  
    {  
     var students = context.Students.ToList();  
    }
  + Common features:
    - Code First, Database First.
    - Lazy Loading and Eager Loading.

## Security and Cryptography

* **Hashing**:
* using System.Security.Cryptography;  
  using System.Text;  
    
  var data = Encoding.UTF8.GetBytes("password");  
  var hash = SHA256.Create().ComputeHash(data);
* **Encryption**:
* using System.Security.Cryptography;  
    
  var aes = Aes.Create();  
  aes.Key = Encoding.UTF8.GetBytes("1234567890123456");  
  aes.IV = Encoding.UTF8.GetBytes("1234567890123456");  
    
  var encryptor = aes.CreateEncryptor();

## Dynamic Type

* Type is resolved at runtime.
* Example:
* dynamic obj = "Hello";  
  Console.WriteLine(obj.Length);  
  obj = 123;  
  Console.WriteLine(obj + 10);

## Database with C# (CRUD)

1. **Connection**:

* using (var connection = new SqlConnection("YourConnectionString"))  
  {  
   connection.Open();  
   Console.WriteLine("Connected");  
  }

1. **CRUD Operations**:

* // Create  
  var insertCommand = new SqlCommand("INSERT INTO Employees (Name) VALUES ('John')", connection);  
  insertCommand.ExecuteNonQuery();  
    
  // Read  
  var selectCommand = new SqlCommand("SELECT \* FROM Employees", connection);  
  using (var reader = selectCommand.ExecuteReader())  
  {  
   while (reader.Read())  
   Console.WriteLine(reader["Name"]);  
  }  
    
  // Update  
  var updateCommand = new SqlCommand("UPDATE Employees SET Name = 'Jane' WHERE Name = 'John'", connection);  
  updateCommand.ExecuteNonQuery();  
    
  // Delete  
  var deleteCommand = new SqlCommand("DELETE FROM Employees WHERE Name = 'Jane'", connection);  
  deleteCommand.ExecuteNonQuery();

## DTO and POCO Models

1. **DTO (Data Transfer Object)**:
   * Used to transfer data between layers or systems.
   * Does not contain any business logic.
   * Example:
   * public class EmployeeDto  
     {  
      public int Id { get; set; }  
      public string Name { get; set; }  
      public string Department { get; set; }  
     }
2. **POCO (Plain Old CLR Object)**:
   * Simple objects with properties, often used with ORM tools like Entity Framework.
   * Example:
   * public class Employee  
     {  
      public int Id { get; set; }  
      public string Name { get; set; }  
      public string Department { get; set; }  
      public DateTime CreatedAt { get; set; }  
     }
   * **Difference between DTO and POCO**:
     + DTO is specifically used for data transfer and may not map directly to the database.
     + POCO often maps directly to database tables and is used in ORM frameworks like EF.