C#, ADO.NET Exercises

1. Write a C# program to Set Up the Development Environment

Objective: Set up your C# development environment.
Instructions:

- Install the latest .NET SDK and Visual Studio or Visual Studio Code.
- Create a new C# console application using Visual Studio or Visual Studio Code.
- Validate the setup by writing and executing a simple “Hello World” program in *Main()*.

2. Write a C# program to Explore Value vs Reference Types

Objective: Understand the differences between value types and reference types.
Instructions:

- Define variables using both value types (e.g., *int, double*) and reference types (e.g., *string*, a custom class).
- Create methods that attempt to modify these variables.
- Print the values before and after method calls to illustrate the behavior of value types and reference types.

3. Write a C# program to Use Primary Constructors in C# 12

Objective: Explore the use of primary constructors in C# 12.
Instructions:

- Create a *Person* class using the primary constructor syntax.
- Instantiate the class and display the person's details using auto-implemented properties.
- Add a method to display the full information of the person.

4. Write a C# program to Demonstrate Type Inference with *var* and *new()*

Objective: Understand the use of type inference in C#.
Instructions:

- Create variables using *var* and *new()* for different types (e.g., *int*, *string*, a custom class).
- Print the types and values of each variable.

- Discuss when type inference is beneficial and when it might affect readability.

5. Write a C# program to Perform Conditional Logic for Grade Calculation

Objective: Use conditional statements to calculate and display grades.
Instructions:

- Accept a score from the user and use *if, else if, else*, and *switch* to determine the grade.
- Use pattern matching in the *switch* statement to handle different score ranges more concisely.

6. Write a C# program to Loop Through an Array with Different Loop Types

Objective: Practice iterating over arrays using various loops.
Instructions:

- Declare an array of integers with at least 5 elements.
- Use a *for, foreach, while*, and *do-while* loop to iterate through the array.
- Include conditions within the loops to skip or stop based on specific values.

7. Write a C# program to Implement Method Overloading

Objective: Demonstrate method overloading.
Instructions:

- Define a method named *CalculateTotal* that accepts different types/number of parameters (e.g., two integers, three doubles).
- Call each version of the method from *Main()* and display the results to demonstrate method overloading.

8. Write a C# program to Use *ref, out*, and *in* Parameters

Objective: Understand and demonstrate *ref, out*, and *in* parameters.
Instructions:

- Create three separate methods: one using *ref*, one using *out*, and one using *in*.
- Observe how each parameter modifier affects the values passed to the method.
- Print values before and after method calls.

9. Write a C# program to Use Local Functions

Objective: Learn how to define and use local functions.
Instructions:

- Write a method called *CalculateFactorial* that uses a local function to perform the factorial calculation.
- Call the local function from within *CalculateFactorial()* and print the result.

10. Write a C# program to Demonstrate OOP Basics with Constructors

Objective: Implement object-oriented concepts using constructors.
Instructions:

- Create a *Car* class with properties for *Make*, *Model*, and *Year*.
- Implement both a **default constructor** and a **parameterized constructor**.
- Create two objects of the class using both constructors and display their values.

11. Write a C# program to Demonstrate Access Modifiers

Objective: Understand access modifiers and their visibility.
Instructions:

- Create a class with *public*, *private*, and *protected* members.
- Access these members from both a derived class and a non-derived class to demonstrate different access levels.

12. Write a C# program to Use Auto-Properties and Backing Fields

Objective: Understand the use of auto-properties and backing fields.
Instructions:

- Create a class *Product* with auto-implemented properties for *Name* and *Price*.
- Add a backing field for *Price* and implement validation to ensure it can't be set to a negative value.
- Display the property values and demonstrate the validation.

13. Write a C# program to Create and Use Records with *init* Properties

Objective: Explore the use of records and *init* properties in C# 9 and 12.
Instructions:

- Define an immutable *Employee* record with *init* properties.
- Instantiate the record and attempt to modify its properties using the with expression to create a modified copy.
- Print the properties to verify that the original record is unchanged.

14. Write a C# program to Demonstrate Inheritance and Method Overriding

Objective: Learn inheritance and method overriding.
Instructions:

- Create a base class *Shape* with a virtual method *Draw().*
- Derive *Circle* and *Rectangle* classes from *Shape*, and override the *Draw()* method.
- Instantiate objects of both *Circle* and *Rectangle*, and call the *Draw()* method.

15. Write a C# program to Differentiate Abstract Classes and Interfaces

Objective: Understand the differences between abstract classes and interfaces.
Instructions:

- Create an abstract class *Vehicle* with an abstract method *Drive().*
- Create an interface *IDrivable* with a method *Start().*
- Implement both the abstract class and interface in a class *Car*.
- Demonstrate polymorphism and method overriding.

16. Write a C# program to Handle Null References Safely

Objective: Safely handle nullable references in C#.
Instructions:

- Create a class *Person* with nullable reference types.
- Use the null-conditional operator (*?.*) and null-coalescing operator (*??*) to safely access object members.
- Demonstrate null checking for nullable references.

17. Write a C# program to Use Null-Conditional Chaining in a Contact App

Objective: Implement null-conditional chaining to avoid null reference exceptions.
Instructions:

- Simulate a contact list using a *Contact* class with properties like *Name* and *PhoneNumber*.
- Safely access a contact's name using null-conditional chaining (*?.*) if the contact or name is null.
- Display the contact's name only if the object and property are not null.

18. Write a C# program to Use the *required* Modifier in C# 12

Objective: Use the *required* modifier to enforce required properties in a class.
Instructions:

- Define a class *Student* with *required* properties.
- Instantiate the class and attempt to leave a required property uninitialized to observe the compiler's behavior.

19. Write a C# program to Work with Lists and Dictionaries

Objective: Understand how to work with *List<T>* and *Dictionary<K, V>* collections.
Instructions:

- Create a *List<string>* and a *Dictionary<int, string>* and populate them with values.
- Use *foreach* loops to iterate over and print the entries.
- Perform basic operations like adding and removing items.

20. Write a C# program to Use LINQ for Filtering and Projection

Objective: Learn how to use LINQ for querying collections.
Instructions:

- Create a list of *Order* objects with properties *OrderId*, *CustomerName*, and *TotalAmount*.
- Use LINQ to filter orders by price and project selected properties into an anonymous type.
- Display the results of the LINQ query.

21. Write a C# program to Use Pattern Matching with *is* and *switch*

Objective: Explore the power of pattern matching with *is* and *switch*.
Instructions:

- Create a method that accepts an object as a parameter.
- Use pattern matching with is to check the type of the object and display appropriate information.

- Use an enhanced *switch* statement to perform operations based on object types.

22. Write a C# program to Create and Deconstruct Tuples

Objective: Learn how to use tuples for returning multiple values.
Instructions:

- Create a method that returns a tuple with two values: an *int* and a *string*.
- In *Main()*, deconstruct the tuple and print the individual values.

23. Write a C# program to Simulate Async File Upload with Exception Handling

Objective: Understand how to work with asynchronous methods and exception handling.
Instructions:

- Create an asynchronous method that simulates a file upload operation by waiting for 3 seconds.
- Return a success message after the delay, and use *try-catch* to handle any exceptions during the upload process.

24. Write a C# program to Serialize and Deserialize JSON Files

Objective: Learn to serialize and deserialize objects using JSON.
Instructions:

- Create a class *User* with properties like *Name*, *Age*, and *Email*.
- Serialize the object to a JSON string and save it to a file.
- Deserialize the JSON back into an object and print its properties.

25. Write a C# program to Use FileStream and MemoryStream

Objective: Practice reading and writing data using *FileStream* and *MemoryStream*.
Instructions:

- Read text from a file using *FileStream* and display the content.
- Write some data to *MemoryStream* and display the number of bytes written.

26. Write a C# program to Demonstrate Race Conditions with Multi-threading

Objective: Learn about race conditions in multi-threaded applications.
Instructions:

- Create a shared counter variable and spawn multiple threads that increment it.
- Use the *lock* statement to manage concurrent access and prevent race conditions.

27. Write a C# program to Simulate and Resolve a Deadlock

Objective: Simulate and resolve a deadlock.
Instructions:

- Create a program with two threads trying to acquire two locks.
- Simulate a deadlock and then resolve it using *Monitor.TryEnter* to prevent indefinite waiting.

28. Write a C# program to Log with *System.Diagnostics.Trace*

Objective: Learn how to log messages using *Trace*.
Instructions:

- Create a logging class that writes log entries to both the console and a file using *Trace.WriteLine*.
- Set up *TextWriterTraceListener* to output the log messages to a file.

29. Write a C# program to Sanitize Input and Prevent XSS

Objective: Sanitize user input to prevent XSS attacks.
Instructions:

- Simulate a user form that accepts input and displays it.
- Sanitize the input using HTML encoding to prevent any potential script injections.

30. Write a C# program to Perform CRUD Operations using ADO.NET

Objective: Learn how to interact with a database using ADO.NET.
Instructions:

- Connect to a local SQL Server instance using *SqlConnection*.
- Perform basic CRUD operations (Insert, Read, Update, Delete) on an *Employees* table using ADO.NET classes like *SqlCommand*, *SqlDataReader*, and *DataAdapter*.