🎓 CS-23: Programming with C#

Prepared By : Lathiya_Harshal.

🟩 1-Mark Questions

Q1. What is the full form of CLR and what is its main function?
Answer:
CLR stands for **Common Language Runtime**.
It is the **execution engine of the .NET Framework**. Its main function is to **manage the execution of .NET programs**, providing key services like **memory management (garbage collection), security, exception handling, and thread management**.

Q2. Define Boxing and Unboxing.
Answer:
Boxing is the process of converting a **value type** (like `int`) to a **reference type** (object).
Unboxing is the reverse process — converting an object **back to a value type**.

Example:

```
int x = 10;

object obj = x;      // Boxing

int y = (int)obj;    // Unboxing
```

🟨 3-Mark Question

Q3. Differentiate between Value Type and Reference Type with examples.
Answer:

Feature	Value Type	Reference Type
Memory Allocation	Stored in Stack	Stored in Heap (reference stored in stack)
Default Behavior	Contains actual data	Contains reference (address) to the data
Examples	`int`, `float`, `bool`, `struct`	`class`, `object`, `string`, `array`
Copying	Creates a copy of data	Copies only the reference (not data)

Code Example:

```
int a = 10;          // Value type

string s = "Hello";  // Reference type
```

🟦 5-Mark Question

Q4. Explain different types of arrays in C# with suitable syntax and examples (One-dimensional, Rectangular, Jagged).
Answer:

In C#, arrays are used to store multiple values in a single variable. There are **three main types of arrays**:

🔹 **1. One-Dimensional Array**

- It is a linear array with elements accessed using a single index.

Syntax:

```
int[] arr = new int[3] {10, 20, 30};
```

Example:

```
for (int i = 0; i < arr.Length; i++) {
    Console.WriteLine(arr[i]);
}
```

🔹 **2. Rectangular Array (Multi-dimensional Array)**

- It is a table-like structure with **rows and columns**.

Syntax:

```
int[,] matrix = new int[2, 3] {
    {1, 2, 3},
    {4, 5, 6}
};
```

Example:

```
for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 3; j++) {
        Console.Write(matrix[i, j] + " ");
    }
    Console.WriteLine();
}
```

🔹 **3. Jagged Array**

- It is an **array of arrays**, meaning rows can have different lengths.

Syntax:

```
int[][] jaggedArr = new int[2][];
jaggedArr[0] = new int[3] {1, 2, 3};
jaggedArr[1] = new int[2] {4, 5};
```

Example:

```
for (int i = 0; i < jaggedArr.Length; i++) {
    for (int j = 0; j < jaggedArr[i].Length; j++) {
        Console.Write(jaggedArr[i][j] + " ");
    }
    Console.WriteLine();
}
```

✅ Unit 2: Class, Inheritance, Property, Indexer, Delegates, Collections

---

🟩 1-Mark Questions

Q1. Define Constructor and mention its types.
Answer:
A **constructor** is a **special method** in a class that is **automatically called when an object is created**. Its purpose is to **initialize the object's data members**.

Types of Constructors in C#:

1. **Default Constructor** – No parameters.
2. **Parameterized Constructor** – Takes arguments.
3. **Copy Constructor** – Initializes using another object.
4. **Static Constructor** – Initializes static data.
5. **Private Constructor** – Restricts object creation from outside.

Q2. What is a Sealed class in C#?
Answer:
A **sealed class** is a class that **cannot be inherited**.
It is declared using the `sealed` keyword. It is used to **prevent further derivation** and improve performance in some cases.

Example:

```
sealed class MyClass {
    public void Display() {
        Console.WriteLine("Sealed Class");
    }
}
```

🟨 3-Mark Question

Q3. Explain the difference between Abstract Class and Interface with one example each.
Answer:

Feature	Abstract Class	Interface
Keyword	`abstract`	`interface`
Members	Can have fields, methods (with/without body)	Only method/property/event declarations
Access Modifiers	Can have them	Cannot have any access modifiers
Inheritance	Supports single inheritance	Supports multiple interfaces

---

Abstract Class Example:

```
abstract class Animal {
    public abstract void Sound();
}

class Dog : Animal {
    public override void Sound() {
        Console.WriteLine("Bark");
    }
}
```

Interface Example:

```
interface IShape {
    void Draw();
}

class Circle : IShape {
    public void Draw() {
        Console.WriteLine("Drawing Circle");
    }
}
```

🟦 5-Mark Question

Q4. Write a program in C# that demonstrates the use of Delegates and Events.
Answer:

- **Delegates** are object-oriented function pointers that allow methods to be called indirectly.
- **Events** are a way for a class to **notify other classes or objects** when something of interest happens.

Program:

```
using System;


// Step 1: Define a delegate

public delegate void Notify();


public class ProcessBusinessLogic {

    // Step 2: Declare an event using the delegate

    public event Notify ProcessCompleted;

    public void StartProcess() {

        Console.WriteLine("Process Started...");

        // Simulate some process

        System.Threading.Thread.Sleep(1000);

        Console.WriteLine("Process Completed.");


        // Step 3: Raise the event

        if (ProcessCompleted != null)

            ProcessCompleted.Invoke();

    }

}


public class Subscriber {

    public void Message() {

        Console.WriteLine("Subscriber: Process Completed Notification Received!");

    }

}


class Program {

    static void Main() {

        ProcessBusinessLogic bl = new ProcessBusinessLogic();

        Subscriber sub = new Subscriber();

        // Step 4: Subscribe to the event

        bl.ProcessCompleted += sub.Message;

        bl.StartProcess();

}

}
```

Explanation:

- `Notify` is a delegate type.
- `ProcessCompleted` is an event of type `Notify`.
- `StartProcess()` raises the event after the process ends.
- `Subscriber` class has a method `Message()` which is **invoked when the event is raised**.

Awesome! Let's dive into **Unit 3: Windows Programming** — here are the very detailed answers for all 1-mark, 3-mark, and 5-mark questions.

✅ Unit 3: Windows Programming

🟩 1-Mark Questions

Q1. Name any four commonly used Windows Form controls.
Answer:
Four commonly used Windows Forms controls are:

1. **Label** – Used to display static text.
2. **TextBox** – Accepts input from the user.
3. **Button** – Triggers an action when clicked.
4. **ComboBox** – Provides a dropdown list of items.

Q2. What is a MessageBox? Mention any two types of `show()` methods.
Answer:
A **MessageBox** is a **popup dialog** used to display a message or prompt to the user. It is modal (blocks other actions until dismissed).

Two types of `Show()` overloads:

1. `MessageBox.Show(string text)`
2. `MessageBox.Show(string text, string caption, MessageBoxButtons buttons, MessageBoxIcon icon)`

Example:

- MessageBox.Show("Hello!");
- MessageBox.Show("Do you want to save changes?", "Confirm", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

🟨 3-Mark Question

Q3. Explain how events are handled in Windows Forms with a button click example.
Answer:
Event Handling in Windows Forms is the mechanism by which user actions (like clicking a button) are captured and responded to by event handlers (methods).

Steps to handle a button click event:

1. Drag and drop a **Button** from the toolbox.
2. Double-click the button to **generate an event handler**.
3. Write logic inside the method.

Example:

- private void btnClickMe_Click(object sender, EventArgs e)

  {

  MessageBox.Show("Button was clicked!");

  }

- Here, `btnClickMe_Click` is the **event handler**.
- `sender` refers to the button object.
- `EventArgs e` contains event-specific data.

When the user clicks the button, this method is triggered automatically.

🟦 5-Mark Question

Q4. Create a basic Windows Form Application in C# with the following: A TextBox, Label, Button – When the button is clicked, display the text from TextBox in Label.
Answer:

This program demonstrates:

- **UI elements**: `TextBox`, `Label`, `Button`
- **Event Handling**: Button click updates the Label

Design Layout (Visual):

Control Type	Name	Purpose
TextBox	`txtInput`	Accepts user input
Label	`lblOutput`	Displays entered text
Button	`btnDisplay`	Triggers the display action

Code Example:

```
using System;
using System.Windows.Forms;

namespace WindowsFormsApp {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void btnDisplay_Click(object sender, EventArgs e) {
            string input = txtInput.Text;
            lblOutput.Text = "You entered: " + input;
        }
    }
}
```

Perfect! Let's now go through **Unit 4: Database Programming with ADO.NET** with very detailed answers for each question — ideal for exam prep.

✅ Unit 4: Database Programming with ADO.NET

🟩 1-Mark Questions

Q1. Name any two ADO.NET Data Providers.
Answer:
ADO.NET supports different data providers to interact with various databases. Two commonly used providers are:

1. **System.Data.SqlClient** – Used for **SQL Server** databases.

2. **System.Data.OleDb** – Used for **Access or Excel** files and other **OLE DB** supported databases.

Q2. What is the role of a DataReader?
Answer:
A **DataReader** provides a **fast, forward-only, read-only** way of retrieving data from a database. It is part of **connected architecture** and works with the `ExecuteReader()` method.

Example:

```
SqlDataReader reader = command.ExecuteReader();

while (reader.Read()) {
    Console.WriteLine(reader["Name"]);
}
```

🟨 3-Mark Question

Q3. Explain the Connected and Disconnected architectures in ADO.NET.
Answer:

🔹 Connected Architecture

- Requires an **open database connection**.
- Uses `SqlCommand` and `SqlDataReader`.
- Faster for **read-only, forward-only** data access.
- **Consumes resources** while connection remains open.

Example:

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

🔸 Disconnected Architecture

- Uses `DataAdapter` and `DataSet`.
- **Does not require continuous connection**.
- Ideal for working with **bulk data and offline operations**.
- Data can be **updated locally** and synced later.

Example:

```
SqlDataAdapter da = new SqlDataAdapter("SELECT * FROM Students", con);
DataSet ds = new DataSet();
da.Fill(ds, "Students");
```

Conclusion:

- Use **Connected** for simple read operations.
- Use **Disconnected** when working with large datasets or when keeping the connection open is not feasible.

🟦 5-Mark Question

Q4. Write a program in C# to connect to a SQL Server database and retrieve records using `SqlDataAdapter` and `DataSet`.
Answer:

✅ This program demonstrates:

- Connection to a database
- Retrieving data using **disconnected architecture**
- Displaying data using a loop

C# Program:

```
using System;

using System.Data;

using System.Data.SqlClient;


class Program {

    static void Main() {

        // Connection string (change according to your SQL Server)

        string connectionString = "Data Source=localhost;Initial Catalog=StudentDB;Integrated Security=True";


        // SQL query to retrieve data

        string query = "SELECT * FROM Students";

// Create connection and adapter

SqlConnection con = new SqlConnection(connectionString);

SqlDataAdapter adapter = new SqlDataAdapter(query, con);


// Create a DataSet

DataSet ds = new DataSet();

try {

    // Fill dataset using adapter

    adapter.Fill(ds, "Students");


    // Accessing DataTable

    DataTable table = ds.Tables["Students"];


    Console.WriteLine("ID\tName\t\tCity");

        Console.WriteLine("---------------------------------");


        // Loop through rows

        foreach (DataRow row in table.Rows) {

            Console.WriteLine($"{row["ID"]}\t{row["Name"]}\t{row["City"]}");

        }

    }

    catch (Exception ex) {

        Console.WriteLine("Error: " + ex.Message);

    }

  }

}
```

Explanation:

- `SqlConnection`: Connects to database
- `SqlDataAdapter`: Executes query and fills data
- `DataSet`: Stores retrieved data
- `DataTable`: Accesses specific table
- Uses **Disconnected** architecture — connection isn't kept open while accessing data

---

✅ Unit 5: User Controls, Crystal Reports, Setup Projects

🟩 1-Mark Questions

Q1. What is a User Control in C#?
Answer:
A **User Control** in C# is a **custom, reusable component** made by combining existing Windows Forms controls (like TextBox, Label, etc.) into one unit.
It behaves like a regular control and can be reused across multiple forms or projects.

Example: A login form (with Username, Password fields) can be built as a User Control and reused wherever needed.

Q2. Mention any two types of Setup Projects.
Answer:
Two common types of Setup Projects used for deploying Windows applications:

1. **Windows Installer Project (.msi)** – Packages application into a single installer.
2. **ClickOnce Deployment** – Allows easy web-based or network-based deployment of Windows Forms apps.

🟨 3-Mark Question

Q3. What are the different sections of a Crystal Report? Explain briefly.
Answer:

A **Crystal Report** consists of the following sections:

1. **Report Header:**
    - Appears **only once at the beginning**.
    - Used for **title, company name, logo**.
2. **Page Header:**
    - Appears **at the top of each page**.
    - Contains **column headings or page titles**.
3. **Details Section:**
    - **Repeats for every record** in the dataset.
    - Core data is displayed here.
4. **Page Footer:**

- Appears **at the bottom of each page**.
- Typically includes **page numbers, print date**.

5. **Report Footer:**
   - Appears **once at the end of the report**.
   - Used for **summary totals, grand totals, signatures**.

🟦 5-Mark Question

Q4. Explain the steps to create and use a Crystal Report in a Windows Form application with one example.
Answer:

Steps to Create and Use Crystal Report in C#

🔹 **Step 1: Add Crystal Report to Project**

- Right-click on project → Add → New Item → Crystal Report
- Choose layout (Standard, Blank, etc.)
- Click "Finish"

🔹 **Step 2: Design the Report**

- Connect to a data source:
  - Use OLE DB or ADO.NET for database connection
- Drag and drop fields (like `Name`, `City`, `Total`) into **Details section**
- Optionally format headers, insert logo, use Summary Fields

🔹 **Step 3: Add a CrystalReportViewer to Form**

- From the toolbox → drag `CrystalReportViewer` to the Windows Form (`Form1.cs`)
- Set its `Dock` property to `Fill`

🔹 **Step 4: Bind Report to Viewer in Code**

```
using CrystalDecisions.CrystalReports.Engine;
using CrystalDecisions.Shared;

public partial class Form1 : Form {
    public Form1() {
        InitializeComponent();
    }

    private void Form1_Load(object sender, EventArgs e) {
        ReportDocument cryRpt = new ReportDocument();
        cryRpt.Load(@"C:\Path\To\YourReport.rpt"); // path to your .rpt file

        // Optional: Set database login info
        cryRpt.SetDatabaseLogon("username", "password", "server", "database");

        crystalReportViewer1.ReportSource = cryRpt;
        crystalReportViewer1.Refresh();
    }
}
```

🔹 **Step 5: Run Application**

- The report will render in the viewer with all fetched data.
- Can be exported to PDF, Excel, or printed directly.

✅ Summary of Key Benefits

- Easy visualization and formatting
- Built-in grouping and summarization
- Drag-drop UI for rapid development
