🎓 CS-23: Programming with C#

Prepared By : Lathiya_Harshal.

Q.1 (a) Attempt the following: (4 Marks)

1. **Full form of CLR:** Common Language Runtime
2. **Array of Array is called:** Jagged array
3. **_____ keyword is used to include namespaces:** `using`
4. **Full form of FCL:** Framework Class Library

---

1. Explain `foreach` loop with example:

The `foreach` loop in C# is used to **iterate over elements in a collection**, such as arrays, lists, or other IEnumerable types. It **automatically fetches each item** one by one without needing an index.

Syntax:

```
foreach (datatype variable in collection) {
    // Code to execute
}
```

Example:

```
string[] fruits = { "Apple", "Banana", "Mango" };

foreach (string fruit in fruits) {
    Console.WriteLine(fruit);
}
```

Output:

```
Apple
Banana
Mango
```

This loop simplifies reading all elements of a collection and avoids out-of-bound errors.

---

2. List types of project in IDE:

In Visual Studio (IDE), different project types can be created based on the application need. Major types include:

1. **Console Application** – Command-line apps (no UI).
2. **Windows Forms Application** – Desktop GUI-based apps.
3. **WPF Application (Windows Presentation Foundation)** – Rich desktop apps with better graphics and UI controls.
4. **ASP.NET Web Application** – Web-based apps.
5. **Class Library** – Reusable code packaged into DLLs.
6. **Setup Project** – For deployment/installer creation.
7. **Windows Service** – Background service-based applications.
8. **Unit Test Project** – For writing test cases.

These project types help organize code based on the target application and deployment requirements.

✅ Q.1 (c) (1): What are Boxing and Unboxing?

📌 **Definition:**

- **Boxing** is the process of **converting a value type** (like `int`, `char`, etc.) **to a reference type** (`object`).
- **Unboxing** is the **reverse process**: converting the **reference type back to a value type**.

📌 **Why it matters:**
Boxing and Unboxing allow value types to be treated as objects, enabling them to work with collections and methods that operate on `object` types.

📌 **Example:**

```
int num = 100;          // Value type
object obj = num;       // Boxing

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

📌 **Explanation:**

- In **Boxing**, the value `100` is copied from `num` to a new object `obj` on the heap.
- In **Unboxing**, the value is cast back to `int`.

📌 **Important Note:**

- **Boxing** involves performance overhead (memory + processing).
- So avoid unnecessary boxing/unboxing in high-performance scenarios.

✅ Q.1 (c) (2): Explain `if` statement with syntax and example

📌 **Definition:** An **`if` statement** in C# is a **conditional control structure** used to **execute code blocks based on a boolean condition** (true or false).

📌 **Syntax:**

```
if (condition) {
    // code to execute if condition is true
}
```

Optional:

```
if (condition) {
    // true block
} else {
    // false block
}
```

📌 **Example:**

```
13. int age = 20;
14.
15. if (age >= 18) {
16.     Console.WriteLine("You are eligible to vote.");
17. } else {
18.     Console.WriteLine("You are not eligible to vote.");
19. }
```

📌 **Output:**

```
20. You are eligible to vote.
```

📌 **Explanation:**

- The condition `age >= 18` is evaluated.
- If true, the first block runs. Otherwise, the `else` block is executed.

✅ Q.1 (d) (1): Explain Architecture of .NET Framework

The **.NET Framework Architecture** is a **platform-independent**, **language-neutral** environment for developing modern applications. It consists of several **core components** that work together.

🔷 Main Components of .NET Framework Architecture:

---

1. Common Language Runtime (CLR):

- **Heart of the .NET Framework**.
- Manages code execution and provides services like:
    - Memory management (Garbage Collection)
    - Security
    - Exception handling
    - Thread management

---

2. Base Class Library (BCL) / Framework Class Library (FCL):

- Predefined **reusable classes**, **interfaces**, and **value types**.
- Includes namespaces like:
    - `System`, `System.IO`, `System.Collections`, `System.Net`, etc.
- Provides support for input/output, file access, data types, strings, etc.

---

3. Common Type System (CTS):

- Ensures all .NET languages (C#, VB.NET, etc.) use **common data types**.
- Promotes **interoperability** between languages.

---

4. Common Language Specification (CLS):

- A set of **basic rules** and standards to be followed by all .NET languages.
- Allows cross-language integration.

---

5. Metadata and Assemblies:

- **Metadata** describes the types and members in code (stored in assemblies).
- **Assemblies** are compiled output in `.exe` or `.dll` format.
- Assemblies contain:
    - IL code (Intermediate Language)
    - Manifest (metadata)
    - Resources

---

6. JIT Compiler (Just-In-Time):

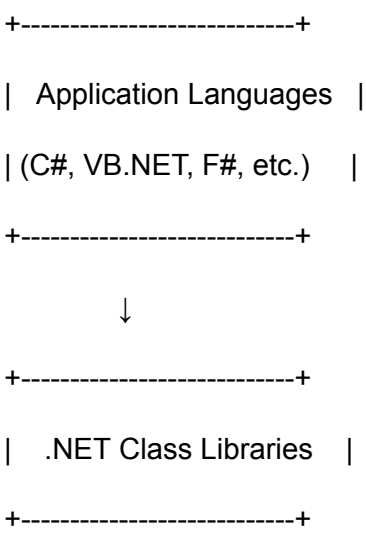
- Converts IL (Intermediate Language) into **native machine code** at runtime.
- Types of JITs:
    - Pre-JIT, Econo-JIT, Normal JIT

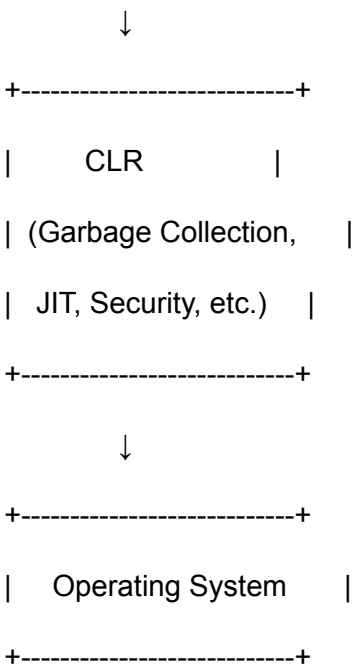
---

7. Language Support:

- .NET supports **multiple languages**, such as:
    - C#, VB.NET, F#, J#, etc.
- All compile to IL and run on CLR.

🔷 Diagram:





✅ Q.1 (d) (2): Explain Data Types in Detail

C# has a rich set of **data types** that are divided into **two major categories**:

🔷 1. Value Types:

- Store the **actual data** directly in memory (stack).
- Examples: `int`, `float`, `char`, `bool`, `struct`, `enum`

Examples:

int x = 10;

char c = 'A';

bool isActive = true;

- Stored on the **stack**.
- Memory-efficient for small values.

🔷 2. Reference Types:

- Store **references (addresses)** to the actual data (which resides in the heap).
- Examples: `string`, `object`, `class`, `interface`, `array`, `delegate`

Examples:

string name = "Mihir";

int[] numbers = {1, 2, 3};

- Stored on the **heap**, reference on **stack**.

🔷 3. Pointer Types (Unsafe Code):

- Direct memory access using pointers.
- Declared with `unsafe` keyword.

```
unsafe {
    int* ptr;
}
```

🔷 4. Nullable Types:

- Allow value types to be assigned `null`.
- Syntax: `int? x = null;`

🔷 5. Dynamic and Var:

- `var`: Implicitly typed local variable (resolved at compile-time).
- `dynamic`: Type resolved at runtime.

```
var age = 25;          // Type inferred as int
dynamic obj = "Hello";   // Type checked at runtime
```

🔷 Comparison Table:

Type	Value Type	Reference Type
Stored in	Stack	Heap
Contains	Data	Reference
Examples	int, char	string, class

✅ Q.2 (a) (1 mark each) — Fill in the blanks (4 Marks)

1. **`sealed`** class will prevent inheritance.
2. **`Enqueue()`** method is used to add value to the end of the Queue.
3. We can use memory through pointer using **`unsafe`** keyword.
4. **`+`** **(plus)** operator is used to add another delegate to multicast delegate.

✅ Q.2 (b) (2 Marks each)

---

1. Give the syntax for defining sealed class

Answer:

A `sealed` class is used to **restrict other classes from inheriting it**.

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

If any class tries to inherit this sealed class, it will cause a **compile-time error**.

---

2. What is method overloading?

Answer:

Method Overloading is a feature in C# that allows **multiple methods with the same name** but **different parameters (type/number/order)** in the **same class**.

It is a form of **compile-time polymorphism**.

Example:

```
class Calculator {
    public int Add(int a, int b) {
        return a + b;
    }

    public double Add(double a, double b) {
        return a + b;
    }

    public int Add(int a, int b, int c) {
        return a + b + c;
    }
}
```

Same method name `Add`, but different signatures.

✅ Q.2 (c) (3 Marks each)

---

1. What is Delegate? Give an Example

Answer:

A **delegate** in C# is like a **function pointer** — it **encapsulates a method** reference and can call that method dynamically.

It allows:

- **Callback functions**
- **Multicasting (multiple methods)**
- **Event handling**

Syntax:

```
public delegate void MyDelegate(); // Step 1: Define
class Test {
    public static void Hello() {
        Console.WriteLine("Hello from delegate!");
    }

    static void Main() {
        MyDelegate del = Hello; // Step 2: Instantiate
        del(); // Step 3: Call method via delegate
    }
}
```

This prints: `Hello from delegate!`

---

2. Differentiate: `ref` vs `out`

Feature	`ref`	`out`
Initialization	Must be initialized before passed	No need to initialize before passing
Usage	Data **in and out** of method	Data is **returned only** from method
Declaration	`ref int x`	`out int x`

Example:

```
void ModifyRef(ref int a) {
    a += 10;
}

void ModifyOut(out int b) {
    b = 20; // Must assign value before using
}
```

✅ Q.2 (d) (5 Marks each)

---

1. What is Collection? Explain any one Collection class in detail

Answer:

A **Collection** in C# is a class used to **store and manipulate groups of related objects**.
They are part of **System.Collections** namespace and can dynamically **grow/shrink in size**.

🔷 Common Collections:

- `ArrayList`
- `Hashtable`
- `Stack`
- `Queue`
- `SortedList`

🔶 Example: `Queue` (FIFO)

- **First-In-First-Out** structure.
- Add item using `Enqueue()`
- Remove item using `Dequeue()`

```
Queue q = new Queue();
q.Enqueue("Apple");
q.Enqueue("Banana");

Console.WriteLine(q.Dequeue()); // Apple
```

Explanation:

- Items are served in the order they were added.
- Useful in printers, task scheduling, etc.

---

2. Explain Single and Multicast Delegate with example

Answer:

🔷 Single Delegate:

- Holds reference to **one method** only.

```
delegate void ShowMessage();

class Test {
    public static void Greet() {
        Console.WriteLine("Hello!");
    }

    static void Main() {
        ShowMessage msg = Greet;
        msg(); // Output: Hello!
    }
}
```

🔷 Multicast Delegate:

- Holds references to **multiple methods**.
- Invokes methods **in order**.

```
delegate void Notify();

class Program {
    static void Show1() => Console.WriteLine("Show 1");
    static void Show2() => Console.WriteLine("Show 2");
    static void Main() {
        Notify n = Show1;
        n += Show2;   // Multicasting
        n();          // Output: Show 1 \n Show 2
    }
}
```

Conclusion:

- Multicast delegate = single delegate + multiple method calls
- Use `+` to combine and `-` to remove from invocation list

✅ Q.3 (a) (1 mark each – 4 Marks)

1. **Default value of interval property of Timer is**: `100` milliseconds
2. **_____ is default property of TextBox control**: `Text`
3. **MDI stands for**: **Multiple Document Interface**
4. **_____ is default event of Timer control**: `Tick`

✅ Q.3 (b) (2 Marks each)

---

1. Difference: TextBox vs RichTextBox

Feature	TextBox	RichTextBox
Text Formatting	Not supported	Supports text formatting (bold, color)
Multiline Support	Partial (needs property set)	Full multiline by default
Use Case	Simple text input	Rich text input/output with formatting

---

2. List out property of Button Control

Important properties of the `Button` control include:

1. **Text** – Sets the caption of the button.
2. **Name** – Identifies the control in code.
3. **Enabled** – Enables or disables the button.
4. **BackColor / ForeColor** – Sets background/text color.
5. **Font** – Sets font style of text.
6. **Size / Location** – Sets dimensions and position.
7. **Visible** – Controls whether the button is visible or hidden.

✅ Q.3 (c) (3 Marks each)

---

1. Explain SaveFileDialog with example

SaveFileDialog is a dialog component that allows users to **select a file name and location to save a file**.

Steps:

- Show dialog using `ShowDialog()`
- Use `FileName` property to get the path

Example:

```
SaveFileDialog saveDlg = new SaveFileDialog();

saveDlg.Filter = "Text files (*.txt)|*.txt|All files (*.*)|*.*";

saveDlg.Title = "Save a File";


if (saveDlg.ShowDialog() == DialogResult.OK) {

    File.WriteAllText(saveDlg.FileName, "Hello Mihir!");

}
```

- The dialog will allow the user to save text to a file.
- `Filter` helps user choose file type.

---

2. Explain PictureBox control

The **PictureBox** control is used to **display images** (like .jpg, .png, .gif) in a Windows Form application.

Key Properties:

- `Image`: Sets the image to display.
- `SizeMode`: Controls how image is stretched or centered.
  - Values: `Normal`, `StretchImage`, `Zoom`, `AutoSize`, `CenterImage`

Example:

```
pictureBox1.Image = Image.FromFile("C:\\images\\photo.jpg");

pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;
```

This loads and stretches the image inside PictureBox.

✅ Q.3 (d) (5 Marks each)

---

1. Explain MessageBox class with all types of show() method

The **MessageBox** class is used to **display pop-up messages** to the user. It's a **modal dialog box**, blocking the rest of the UI until closed.

Overloads of `MessageBox.Show()` method:

- **1. Basic Message**

```
MessageBox.Show("Hello, User!");
```

- **2. With Title**

```
MessageBox.Show("Welcome!", "Greeting");
```

- **3. With Buttons**

```
MessageBox.Show("Do you want to save?", "Save File", MessageBoxButtons.YesNo);
```

- **4. With Icons**

```
MessageBox.Show("Error occurred", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
```

- **5. With Default Button Selection**

```
MessageBox.Show("Are you sure?", "Exit", MessageBoxButtons.YesNoCancel, MessageBoxIcon.Question,
MessageBoxDefaultButton.Button2);
```

Common `MessageBoxButtons`:

- `OK`, `OKCancel`, `YesNo`, `YesNoCancel`, `RetryCancel`, `AbortRetryIgnore`

Common `MessageBoxIcon`:

- `Information`, `Warning`, `Error`, `Question`

---

2. Explain RadioButton and CheckBox controls with example

- **RadioButton**
  - Allows selection of **only one option** from a group.
  - Usually grouped inside a `GroupBox`.

```
if (radioButtonMale.Checked)

    MessageBox.Show("Gender: Male");
```

- **CheckBox**
  - Allows **multiple selections**.

- Can be **checked or unchecked independently**.

```
if (checkBoxJava.Checked)
    MessageBox.Show("You selected Java");
```

Design Example:

```
[ ] Java     [✓] C#        ← CheckBoxes (multiple selections)
(•) Male     ( ) Female    ← RadioButtons (single selection)
```

✅ Q.4 (a) (1 Mark each – 4 Marks)

1. **ADO stands for: ActiveX Data Objects**
2. **DataReader is a Disconnected Architecture component: FALSE**
   (DataReader is used in Connected Architecture)
3. **To connect with SQL Server _____ namespace is used:** `System.Data.SqlClient`
4. **Fill method is used with DataReader: FALSE**
   (Fill() is used with DataAdapter, not DataReader)

✅ Q.4 (b) (2 Marks each)

---

1. Explain Command Object

Answer:

The **Command object** (`SqlCommand`) in ADO.NET is used to **execute SQL queries or stored procedures** on a database.

Key Properties:

- `CommandText` – SQL query
- `Connection` – Connection object
- `CommandType` – Text or StoredProcedure

Example:

SqlCommand cmd = new SqlCommand("SELECT * FROM Students", con);

You can execute commands using:

- `ExecuteReader()` – for SELECT queries
- `ExecuteNonQuery()` – for INSERT/UPDATE/DELETE
- `ExecuteScalar()` – to return single value

2. What is DataTable?

Answer:

A **DataTable** is an in-memory **representation of a table of data**.

- Belongs to `System.Data` namespace
- Used in **disconnected architecture**
- Can be part of a **DataSet** or used independently

Key Features:

- Stores rows and columns
- Can define primary key, constraints, data types

Example:

```
DataTable dt = new DataTable();

dt.Columns.Add("ID");

dt.Columns.Add("Name");
```

✅ Q.4 (c) (3 Marks each)

---

1. Explain DataReader with example

Answer:

`SqlDataReader` is used to **read data in a forward-only, read-only** manner from a database using **Connected Architecture**.

Steps:

1. Open connection
2. Create command
3. Use `ExecuteReader()`
4. Loop through `Read()`

Example:

```
SqlConnection con = new SqlConnection("connection_string");

SqlCommand cmd = new SqlCommand("SELECT * FROM Students", con);

con.Open();
```

```
SqlDataReader dr = cmd.ExecuteReader();
while (dr.Read()) {
    Console.WriteLine(dr["Name"]);
}
con.Close();
```

---

2. Difference between `ExecuteNonQuery()` and `ExecuteScalar()`

Feature	`ExecuteNonQuery()`	`ExecuteScalar()`
Return Type	Integer (no. of rows affected)	Single value (first column of first row)
Usage	INSERT, UPDATE, DELETE	SELECT (aggregate values like COUNT)
Example Output	1 (1 row affected)	120 (Total Students)

Examples:

```
// Insert
cmd.CommandText = "INSERT INTO Students VALUES('Mihir', 'IT')";
int rows = cmd.ExecuteNonQuery();  // returns number of rows affected

// Count
cmd.CommandText = "SELECT COUNT(*) FROM Students";
int count = (int)cmd.ExecuteScalar();  // returns single value
```

✅ Q.4 (d) (5 Marks each)

---

1. Explain the Architecture of ADO.NET

Answer:

ADO.NET provides a bridge between your C# application and a data source (like SQL Server).

Architecture is divided into two models:

🔷 **1. Connected Model**

- Always connected to DB
- Uses:

- `SqlConnection`
- `SqlCommand`
- `SqlDataReader`

🔷 **2. Disconnected Model**

- Data is retrieved, then connection is closed
- Uses:
  - `SqlDataAdapter`
  - `DataSet`, `DataTable`

Diagram:

Application

↓

SqlConnection ↔ SqlCommand

↓ ↓

SqlDataReader SqlDataAdapter ↔ DataSet ↔ DataTable

Key Benefits:

- Disconnected model improves performance and scalability
- Connected model is better for real-time operations

---

2. Explain Connection and DataAdapter object in detail

🔹 **SqlConnection**

- Manages **database connectivity**
- Needs connection string (server, database, auth info)

```
SqlConnection con = new SqlConnection("your_connection_string");
con.Open();
```

🔹 **SqlDataAdapter**

- Acts as a **bridge between database and DataSet**
- Uses Fill() to load data and Update() to save changes

```
SqlDataAdapter da = new SqlDataAdapter("SELECT * FROM Students", con);

DataSet ds = new DataSet();

da.Fill(ds, "Students");
```

✅ Q.5 (a) (1 Mark each – 4 Marks)

1. **_____ control is used on form to view the Crystal Report:** `CrystalReportViewer`

2. **By default, how many sections are provided in Crystal Report?** `5`
   (Report Header, Page Header, Details, Page Footer, Report Footer)

3. **_____ field of Crystal Report is used to create Formula:** `Formula Field`

4. **Drop Down is a type of Crystal Report: FALSE**

✅ Q.5 (b) (2 Marks each)

---

1. List out types of Reports

Answer:

There are several types of reports in **Crystal Reports**, based on structure and data source:

1. **Standard Report** – Basic report using a database/table.
2. **Cross-Tab Report** – Displays summarized data in matrix form (rows vs columns).
3. **Mail Label Report** – Used for printing addresses or labels.
4. **Drill Down Report** – Allows detailed view when clicking on summary data.
5. **Subreport** – Embeds one report within another.
6. **Linked Report** – Two reports with common fields/data source.

---

2. Write the name of any four Special Fields of Crystal Report

Answer:

Special fields are built-in fields that insert useful metadata in the report.

✔️ Four common **Special Fields**:

1. **Print Date** – Displays the current print date.
2. **Page Number** – Shows current page number.
3. **Total Page Count** – Total number of pages in report.
4. **Report Title** – Displays the title of the report.

Other examples: File Path, Author, Modification Date, Data Source

✅ Q.5 (c) (3 Marks each)

---

1. Explain types of Setup Projects

Answer:

Setup projects are used to **deploy applications to end users**. Types include:

1. **Windows Installer (.msi)**
    - Traditional setup file for installing on Windows.
    - Created using Visual Studio Setup Project.
2. **ClickOnce Deployment**
    - Web-based or network-based deployment.
    - Simplifies updating apps automatically.
3. **MSIX Packaging (new)**
    - Modern packaging format for secure and reliable deployment.

---

2. Explain Summary Field in Crystal Report

Answer:

A **Summary Field** is used to perform **aggregations** like:

- **SUM, AVG, COUNT, MAX, MIN**
- Applied to numeric or grouped data fields

Usage:

- Go to **Field Explorer** → Right-click on field → Insert Summary
- Select **operation** (Sum, Count, etc.) and **group** if needed

Example:

- Show **total sales amount** for each customer:

Customer Name	Total Sales
John	$1200
Jane	$800

Summary fields help in creating dashboards, analytics, or financial reports.

✅ Q.5 (d) (5 Marks each)

---

1. Explain types of Reports in detail

Answer:

Crystal Reports offers various types depending on the reporting requirement:

🔹 **1. Standard Report**

- Basic report layout with rows/columns
- Supports grouping, filtering, formatting

🔹 **2. Cross-Tab Report**

- Displays summary data in **matrix** format (rows vs columns)
- Useful for analysis like "Product-wise Monthly Sales"

🔹 **3. Mail Label Report**

- Designed for printing mailing labels
- Based on contact/address data

🔹 **4. Drill-Down Report**

- User clicks on a summary value to **view detailed breakdown**
- Makes reports **interactive**

🔹 **5. Subreport**

- A report inside another report
- Ideal for related data from **different tables** or **different data sources**

🔹 **6. Linked Report**

- Allows sharing data across multiple reports using common fields

---

Summary:

Report Type	Purpose
Standard	Regular data representation
Cross-Tab	Matrix view for grouped summary
Mail Label	Address labels for printing
Subreport	Nesting reports
Drill-Down	Detailed view on click

---

2. Write steps to create Crystal Report

Answer:

Here are the step-by-step instructions to create a Crystal Report in Visual Studio:

🔷 Step 1: Add Report File

- Right-click project → Add → New Item → Select **Crystal Report (.rpt)**
- Choose layout (Standard, Blank Report, etc.)

🔷 Step 2: Connect to Data Source

- Use **Database Expert** to:
    - Select **data source** (e.g., SQL Server, Oracle, Access)
    - Choose tables or views

🔷 Step 3: Design the Report

- Drag-and-drop fields from Field Explorer into:
    - **Report Header** (title, logo)
    - **Page Header** (column headings)
    - **Details** (record data)

🔷 Step 4: Apply Formatting & Grouping

- Group data (e.g., by City or Department)
- Use **Summary Fields** (e.g., total salary)
- Insert **Special Fields** (Page Number, Date)

🔷 Step 5: Add CrystalReportViewer on Form

- From Toolbox → Drag `CrystalReportViewer` on a Windows Form

🔷 Step 6: Write Code to Load Report

```
ReportDocument rpt = new ReportDocument();
rpt.Load("Path\\YourReport.rpt");
crystalReportViewer1.ReportSource = rpt;
```

🔷 Step 7: Run the App

- View the report with live data.
- Optionally export to PDF, Excel, etc.
