**Class Assignment 1**

**Name-Utkarsh Sec- Mca(C) Roll-48**

**1.Imagine you are explaining the .NET framework architecture to a colleague unfamiliar with the framework. How would you break down the architecture and its components, such as the CLR, FCL, and the application domains? Provide a structured explanation.**

The .NET Framework is a software development platform by Microsoft that provides a managed environment for building, running, and deploying applications.

The architecture is mainly divided into three key parts:

1. Common Language Runtime (CLR):
   * It is the execution engine of the .NET framework.
   * It handles memory management, thread execution, exception handling, and garbage collection.
   * In short, it ensures that the code runs safely and efficiently.
2. Framework Class Library (FCL):
   * This is a collection of reusable classes, interfaces, and value types that developers can use directly in their code.
   * It contains libraries for tasks like file handling, database operations, networking, and collections.
3. Application Domains:
   * These are isolated environments where .NET applications run.
   * They allow multiple applications to run on the same machine without affecting each other.

Example:  
If we run multiple .NET programs in Visual Studio, each one runs in its own application domain under the CLR, using classes from the FCL.

**2.In a team meeting, you are asked to explain key .NET framework runtime concepts like the Common Language Runtime (CLR), Common Type System (CTS), and Common Language Specification (CLS). How would you present these to ensure clarity and relevance to the team's work?**

1. **Common Language Runtime (CLR):**
   * The CLR is the core runtime environment.
   * It converts Intermediate Language (IL) code into native machine code.
   * It also provides memory management, security, and error handling.
2. **Common Type System (CTS):**
   * CTS defines how data types are declared and used in .NET.
   * It ensures that different languages (like C#, VB.NET) can interact with each other easily.
   * For example, an int in C# and an Integer in VB.NET represent the same data type in CTS.
3. **Common Language Specification (CLS):**
   * CLS defines a set of basic rules that all .NET languages must follow to ensure compatibility.
   * This allows code written in one language to be used in another.

**Example:**  
If we write a class in C# and another in VB.NET, both can work together because they follow the same CTS and CLS rules.

**3.You are developing a large-scale application and need to explain to a junior developer how assemblies are used in .NET framework to organize and deploy the application. Provide an explanation of assemblies and include an example scenario where multiple assemblies are used.**

Assemblies are the building blocks of a .NET application.  
They are compiled units that contain the code and metadata required to run an application.

Assemblies can be of two types:

1. **Private Assembly:** Used by a single application.
2. **Shared Assembly:** Used by multiple applications (stored in the Global Assembly Cache – GAC).

Each assembly contains:

* **Manifest** (metadata about the assembly)
* **MSIL Code** (compiled Intermediate Language code)
* **Type metadata and resources**

**Example:**  
Suppose we build a hospital management system with different modules:

* Hospital.Core.dll for common classes
* Hospital.Patient.dll for patient details
* Hospital.Billing.dll for billing functions  
  Each module can be compiled as a separate assembly and reused in other applications.

**4.In your project, you notice a developer struggling to organize classes and methods properly. How would you explain the concept of namespaces in .NET framework and demonstrate how they are used to avoid naming conflicts in large projects?**

A **namespace** is used to organize classes, methods, and other members logically to avoid naming conflicts in large projects. It acts like a container for classes.

using System;

namespace CollegeApp

{

class Student

{

public void Display()

{

Console.WriteLine("This is the Student class");

}

}

}

namespace CollegeApp.Teachers

{

class Teacher

{

public void Show()

{

Console.WriteLine("This is the Teacher class");

}

}

}

Here, both Student and Teacher classes are under different namespaces, avoiding confusion.  
Namespaces make code more organized and maintainable.

**5.During a code review, a developer confuses primitive types with reference types in their application. How would you explain the difference between primitive types and reference types?**

| **Feature** | **Primitive (Value) Types** | **Reference Types** |
| --- | --- | --- |
| **Stored in** | Stack memory | Heap memory |
| **Contains** | Actual data | Reference (address) to the data |
| **Examples** | int, float, bool, char | string, array, class, object |
| **Copied behavior** | Copies the actual value | Copies the reference (both point to same memory) |

**Code:-**

int a = 10;

int b = a; // Value copy

string s1 = "Hello";

string s2 = s1; // Reference copy

If we change b, a remains the same (value type).  
If we change s2, both s1 and s2 may refer to the same object (reference type).

**6.While refactoring a piece of C# code, you notice both value types and reference types are being used incorrectly. Explain the difference between value types and reference types in C#, and provide examples to clarify their behaviour in memory .**

In C#, **value types** store data directly in memory, while **reference types** store a reference (memory address) to the actual data.

* Value types are stored on the **stack**.
* Reference types are stored on the **heap**.

**Examples of Value Types:** int, float, bool, char, struct  
**Examples of Reference Types:** class, string, array, object

**Example Code:**

using System;

namespace Assignment\_1

{

internal class Ques6

{

static void Main()

{

int x = 10; // Value type

int y = x;

y = 20;

string s1 = "Utkarsh"; // Reference type

string s2 = s1;

s2 = "Hi";

Console.WriteLine("x = " + x); // Output: 10

Console.WriteLine("y = " + y); // Output: 20

Console.WriteLine("s1 = " + s1); // Output: Utkarsh

Console.WriteLine("s2 = " + s2); // Output: Hi

}

}

}

**7.You are tasked with creating a method that demonstrates both implicit and explicit type conversions. Write a program in C# that converts an int to a double implicitly and a double to an int explicitly, explaining each step in your code.**

Implicit Conversion: Done automatically by the compiler (safe conversion).

Explicit Conversion: Requires casting by the programmer (possible data loss).

**Example Code:**

using System;

namespace Assignment\_1

{

internal class Ques7

{

static void Main()

{

int num = 10; // Integer

double d = num; // Implicit conversion (int → double)

double val = 9.78;

int n = (int)val; // Explicit conversion (double → int)

Console.WriteLine("Implicit Conversion: " + d); // Output: 10

Console.WriteLine("Explicit Conversion: " + n); // Output: 9

}

}

}

**8.A junior developer asks for help writing a program to determine whether a number is positive, negative, or zero. Use if-else statements to write this program in C#, and explain the logic behind the code.**

We can use **if–else statements** to determine whether a number is positive, negative, or zero.

**Example Code:**

using System;

namespace Assignment\_1

{

internal class Ques8

{

static void Main()

{

Console.Write("Enter a number: ");

int num = Convert.ToInt32(Console.ReadLine());

if (num > 0)

Console.WriteLine("Number is Positive");

else if (num < 0)

Console.WriteLine("Number is Negative");

else

Console.WriteLine("Number is Zero");

}

}

}

**9.You are explaining control flow constructs to a new hire. Use a switch-case construct to explain how it works in C#. Illustrate the use of this construct by writing a program that takes a number (1-5) and prints the corresponding weekday.**

A switch-case statement is used when we have multiple possible outcomes based on one condition.

using System;

namespace Assignment\_1

{

internal class Ques9

{

static void Main()

{

Console.Write("Enter a number (1-5): ");

int day = Convert.ToInt32(Console.ReadLine());

switch (day)

{

case 1: Console.WriteLine("Monday"); break;

case 2: Console.WriteLine("Tuesday"); break;

case 3: Console.WriteLine("Wednesday"); break;

case 4: Console.WriteLine("Thursday"); break;

case 5: Console.WriteLine("Friday"); break;

default: Console.WriteLine("Invalid input"); break;

}

}

}

}

**Explanation:**

* The switch-case matches the input with the correct case and executes that block.
* If none match, default is executed.

**10.You are mentoring a developer on decision constructs in C#. Demonstrate how to use nested if-else and switch-case statements together by writing a program that checks a number and prints whether it is even/odd and whether it falls into specific ranges (e.g., 0-10, 11-20).**

This program checks whether a number is **even or odd**, and then tells which range it falls in.

**Example Code:**

using System;

namespace Assignment\_1

{

internal class Ques10

{

static void Main()

{

Console.Write("Enter a number: ");

int num = Convert.ToInt32(Console.ReadLine());

// Check even or odd

if (num % 2 == 0)

Console.WriteLine("Even Number");

else

Console.WriteLine("Odd Number");

// Range Check using switch

switch (num)

{

case int n when (n >= 0 && n <= 10):

Console.WriteLine("Range: 0–10");

break;

case int n when (n >= 11 && n <= 20):

Console.WriteLine("Range: 11–20");

break;

default:

Console.WriteLine("Out of Range");

break;

}

}

}

}

**Example Output:**

Enter a number: 14

Even Number

Range: 11–20

**11.During a live coding session, you are asked to write a program that prints the Fibonacci series using a for loop in C#. Provide a detailed explanation of your approach, and explain how the loop is used to generate the series.**

The **Fibonacci series** is a sequence where each number is the sum of the previous two numbers.  
Example: 0, 1, 1, 2, 3, 5, 8, 13…

**Example Code:**

using System;

namespace Assignment\_1

{

internal class Ques11

{

static void Main()

{

int n1 = 0, n2 = 1, n3, i, number;

Console.Write("Enter the number of terms: ");

number = Convert.ToInt32(Console.ReadLine());

Console.Write(n1 + " " + n2 + " ");

for (i = 2; i < number; i++)

{

n3 = n1 + n2;

Console.Write(n3 + " ");

n1 = n2;

n2 = n3;

}

}

}

}

**12.You are leading a training session on loops in C#. Explain the key differences between while and do-while loops, and provide examples of each where one might be more appropriate than the other.**

| **Feature** | **while loop** | **do-while loop** |
| --- | --- | --- |
| **Condition check** | Before execution | After execution |
| **Executes at least once?** | No | Yes |
| **Syntax** | while(condition) | do { } while(condition); |

**Example of while loop:**

using System;

namespace Assignment\_1

{

internal class Ques12

{

static void Main()

{

int i = 1;

while (i <= 5)

{

Console.WriteLine("While Loop Count: " + i);

i++;

}

}

}

}

**Example of do-while loop:**

int j = 1;

do

{

Console.WriteLine("Do-While Loop Count: " + j);

j++;

} while (j <= 5);

**Explanation:**

* The while loop checks the condition **before** execution.
* The do-while loop runs **at least once**, even if the condition is false.

**13.You are developing a pattern generation tool for a project. Write a program in C# that uses nested loops to generate a pyramid pattern of stars (\*). Explain how the loops work together to produce the pattern.**

using System;

namespace Assignment\_1

{

internal class Ques13

{

static void Main()

{

Console.Write("Enter number of rows: ");

int rows = Convert.ToInt32(Console.ReadLine());

for (int i = 1; i <= rows; i++)

{

for (int j = 1; j <= rows - i; j++)

{

Console.Write(" ");

}

for (int k = 1; k <= (2 \* i - 1); k++)

{

Console.Write("\*");

}

Console.WriteLine();

}

}

}

}

**Output Example (for 5 rows):**

\*

\*\*\*

\*\*\*\*\*

\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*

**14.You are giving a presentation on object-oriented programming. Define Encapsulation, Inheritance, Polymorphism, and Abstraction, and provide real-world examples of each in the context of C# development.**

**Encapsulation:**

* Wrapping data and methods into a single unit (class).
* Example: A class BankAccount keeps account balance private and allows deposit/withdraw through methods.

class BankAccount

{

private double balance;

public void Deposit(double amount) => balance += amount;

public double GetBalance() => balance;

}

**Inheritance:**

* When one class derives from another.
* Example: Car class inherits from Vehicle.

class Vehicle { public void Start() => Console.WriteLine("Vehicle started"); }

class Car : Vehicle { public void Horn() => Console.WriteLine("Car horn blown"); }

**Polymorphism:**

* Same method behaves differently based on the object.
* Example:

class Shape { public virtual void Draw() => Console.WriteLine("Drawing Shape"); }

class Circle : Shape { public override void Draw() => Console.WriteLine("Drawing Circle"); }

**Abstraction:**

* Showing only essential features and hiding details.
* Example:

abstract class Animal

{

public abstract void Sound();

}

class Dog : Animal

{

public override void Sound() => Console.WriteLine("Bark");

}

**Summary:**  
Encapsulation = data hiding  
Inheritance = code reuse  
Polymorphism = many forms  
Abstraction = hiding complexity

**15.In a team discussion, you are asked to demonstrate the use of constructors and destructors in C#. Write a C# program that includes both, explaining the lifecycle of an object from creation to destruction.**

using System;

namespace Assignment\_1

{

internal class Ques15

{

class Student

{

public Student() // Constructor

{

Console.WriteLine("Constructor called: Object Created");

}

~Student() // Destructor

{

Console.WriteLine("Destructor called: Object Destroyed");

}

}

static void Main()

{

Student s = new Student();

Console.WriteLine("Inside Main Method");

}

}

}

**16.A team member is confused about access modifiers in C#. How would you explain public, private, protected, and internal modifiers, and demonstrate their use by writing a small C# class with methods using different access levels?**

Access modifiers define the **visibility** or **scope** of a class or its members.  
C# provides **five main access modifiers:**

| **Modifier** | **Accessible From** | **Example** |
| --- | --- | --- |
| public | Anywhere | public int age; |
| private | Within the same class | private int salary; |
| protected | In the same class and derived classes | protected void Display(); |
| internal | Within the same assembly/project | internal string name; |
| protected internal | In same assembly or derived class | Used for shared + inherited access |

using System;

namespace Assignment\_1

{

internal class Ques16

{

class Employee

{

public string Name = "Utkarsh";

private double Salary = 50000;

protected int ID = 48;

internal string Department = "IT";

public void ShowDetails()

{

Console.WriteLine($"Name: {Name}, Salary: {Salary}, ID: {ID}, Department: {Department}");

}

}

class Manager : Employee

{

public void ShowProtected()

{

Console.WriteLine("Accessing Protected ID: " + ID);

}

}

static void Main()

{

Employee e = new Employee();

e.ShowDetails();

Manager m = new Manager();

m.ShowProtected();

}

}

}

Output:

Name: Utkarsh, Salary: 50000, ID: 48, Department: IT

Accessing Protected ID: 48

**17.You are tasked with illustrating the concept of inheritance in C#. Write a program where a Vehicle class is inherited by a Car class and a Bike class, each with their own unique methods. Demonstrate how inheritance allows code reuse.**

**Single Inheritance:** One class inherits another.  
**Multilevel Inheritance:** A class inherits from a derived class.

**Example Code:**

using System;

namespace Assignment\_1

{

internal class Ques17

{

class Animal

{

public void Eat() => Console.WriteLine("Eating...");

}

class Dog : Animal

{

public void Bark() => Console.WriteLine("Barking...");

}

class Puppy : Dog

{

public void Sleep() => Console.WriteLine("Sleeping...");

}

static void Main()

{

Puppy p = new Puppy();

p.Eat(); // from Animal

p.Bark(); // from Dog

p.Sleep(); // from Puppy

}

}

}

**Output:**

**Eating...**

**Barking...**

**Sleeping...**

**18.In a bug-fixing scenario, your team needs to handle unexpected runtime errors. Explain how the try-catch-finally blocks work in C# with an example of catching and handling an arithmetic exception, and how finally is always executed.**

1. **Method Overloading** → Compile-time polymorphism (same method name, different parameters).
2. **Method Overriding** → Run-time polymorphism (derived class changes base class method behavior).

**Example Code:**

using System;

namespace Assignment\_1

{

internal class Ques18

{

class Calculator

{

// Overloading

public int Add(int a, int b) => a + b;

public double Add(double a, double b) => a + b;

}

class Parent

{

public virtual void Show() => Console.WriteLine("Parent Class Show()");

}

class Child : Parent

{

public override void Show() => Console.WriteLine("Child Class Show()");

}

static void Main()

{

Calculator c = new Calculator();

Console.WriteLine("Add (int): " + c.Add(5, 10));

Console.WriteLine("Add (double): " + c.Add(2.5, 3.5));

Parent p = new Parent();

p.Show();

Child ch = new Child();

ch.Show();

}

}

}

**Output:**

**Add (int): 15**

**Add (double): 6**

**Parent Class Show()**

**Child Class Show()**

**19.You are implementing a custom exception for a specific error scenario in your application. Write a C# program that demonstrates exception handling by throwing and catching a custom exception, explaining why custom exceptions are beneficial.**

using System;

namespace Assignment\_1

{

internal class Ques19

{

static void Main()

{

try

{

Console.Write("Enter a number: ");

int num = Convert.ToInt32(Console.ReadLine());

int result = 100 / num;

Console.WriteLine("Result: " + result);

}

catch (DivideByZeroException)

{

Console.WriteLine("Error: Cannot divide by zero!");

}

catch (FormatException)

{

Console.WriteLine("Error: Invalid number entered!");

}

finally

{

Console.WriteLine("Finally block executed.");

}

}

}

}

**Output Example:**

Enter a number: 0

Error: Cannot divide by zero!

Finally block executed.

**20.During a code quality meeting, you are asked to highlight the advantages of using exception handling in C#. Explain how proper exception handling improves application’s robustness.**

using System;

namespace Assignment\_1

{

internal class Ques20

{

class InvalidAgeException : Exception

{

public InvalidAgeException(string message) : base(message) { }

}

static void Main()

{

try

{

Console.Write("Enter age: ");

int age = Convert.ToInt32(Console.ReadLine());

if (age < 18)

throw new InvalidAgeException("Age must be 18 or above to vote!");

Console.WriteLine("You are eligible to vote.");

}

catch (InvalidAgeException ex)

{

Console.WriteLine("Custom Exception: " + ex.Message);

}

finally

{

Console.WriteLine("Program finished.");

}

}

}

}

**Output Example:**

**Enter age: 15**

**Custom Exception: Age must be 18 or above to vote!**

**Program finished.**