Capgemini training – Assignment 1(Day 2)

```
using System;
class Program
{
    static void Main()
    {
        Exercise1();
        Exercise2();
        Exercise3();
        Exercise4();
        Exercise5();
        Exercise6();
        Exercise7();
        Exercise8();
        Exercise9();
        Exercise10();
    }

    // Exercise 1: Student Attendance & Eligibility
    static void Exercise1()
    {
        Console.Write("Enter classes attended: ");
        int attended = int.Parse(Console.ReadLine());
        Console.Write("Enter total classes: ");
        int total = int.Parse(Console.ReadLine());

        double percentage = (double)attended / total * 100;
        int displayPercentage = (int)Math.Round(percentage);
        Console.WriteLine($"Attendance Percentage: {displayPercentage}%");
```

Q) Why use rounding instead of truncation?

A) Truncation simply removes the decimal part and can under-represent attendance (e.g., 74.9% → 74%). Rounding gives a more accurate and fair representation (74.9% → 75%). Since eligibility decisions depend on percentage, rounding avoids unintended disqualification.

}

```
// Exercise 2: Online Examination Result Processing
static void Exercise2()
{
    Console.Write("Enter marks 1: ");
    int m1 = int.Parse(Console.ReadLine());

    Console.Write("Enter marks 2: ");
    int m2 = int.Parse(Console.ReadLine());

    Console.Write("Enter marks 3: ");
    int m3 = int.Parse(Console.ReadLine());

    double average = (m1 + m2 + m3) / 3.0;
    Console.WriteLine($"Average: {average:F2}");
    int scholarshipScore = (int)Math.Round(average);
    Console.WriteLine($"Scholarship Score: {scholarshipScore}");
```

Q) Why does precision loss occur when converting double to int?

A) A double can store fractional values, but int cannot. When converting, the decimal portion is either truncated or rounded, causing loss of precision. This is acceptable for scholarship eligibility where only a whole-number cutoff is required, but not for detailed score analysis.

}

```
// Exercise 3: Library Fine Calculation
static void Exercise3()
{
    Console.Write("Enter fine per day: ");
    decimal finePerDay = decimal.Parse(Console.ReadLine());
    Console.Write("Enter days overdue: ");
    int daysLate = int.Parse(Console.ReadLine());
    decimal totalFine = finePerDay * daysLate;
    double analyticsFine = (double)totalFine;
    Console.WriteLine($"Total Fine: {totalFine}");
    Console.WriteLine($"Logged Fine (double): {analyticsFine}");
```

Q) Why use decimal for fine calculation?

A) decimal provides higher precision for monetary values and avoids rounding errors.

Q) Why convert to double for analytics?

A) Analytics systems favor double due to faster computation and lower memory usage. Minor precision loss is acceptable for trend analysis but not for billing.

```
}
// Exercise 4: Banking Interest Calculation
static void Exercise4()
{
    Console.Write("Enter account balance: ");
    decimal balance = decimal.Parse(Console.ReadLine());
    Console.Write("Enter interest rate (%): ");
    float interestRate = float.Parse(Console.ReadLine());
    decimal interest = balance * (decimal)interestRate / 100;
    balance += interest;
```

```
    Console.WriteLine($"Updated Balance: {balance}");
```

Q) Why can't float be implicitly converted to decimal?
A) float is a binary floating-point type, while decimal is a base-10 type. Implicit conversion could introduce hidden precision errors, so C# forces explicit casting to make the developer aware of potential data loss.

```
}

// Exercise 5: E-Commerce Order Pricing
static void Exercise5()
{
    Console.Write("Enter cart total: ");
    double cartTotal = double.Parse(Console.ReadLine());

    Console.Write("Enter tax rate (decimal): ");
    decimal taxRate = decimal.Parse(Console.ReadLine());
    Console.Write("Enter discount: ");
    decimal discount = decimal.Parse(Console.ReadLine());

    decimal finalAmount = (decimal)cartTotal +
        ((decimal)cartTotal * taxRate) – discount;
    Console.WriteLine($"Final Payable Amount: {finalAmount}");
```

Q) Why convert double to decimal before final calculation?
A) double can produce rounding errors in financial calculations (e.g., 0.1 + 0.2 ≠ 0.3 exactly). Converting to decimal ensures accurate tax and discount calculations, which is critical for financial correctness.

```
}

// Exercise 6: Weather Monitoring
static void Exercise6()
{
```

```
    Console.Write("Enter temperature in Kelvin: ");
    short sensorReading = short.Parse(Console.ReadLine());

    double celsius = sensorReading - 273.15;
    int displayTemp = (int)Math.Round(celsius);

    Console.WriteLine($"Temperature in Celsius: {displayTemp}°C");
```

Q) Why be cautious when converting short to double and then to int?
A) Although short safely converts to double, converting the result to int can cause overflow or data loss if values exceed the int range. Rounding is used to preserve correctness while displaying sensor data.

```
}

// Exercise 7: University Grading Engine
static void Exercise7()
{
    Console.Write("Enter final score: ");
    double finalScore = double.Parse(Console.ReadLine());
    byte grade;
    if (finalScore >= 90) grade = 10;
    else if (finalScore >= 80) grade = 9;
    else if (finalScore >= 70) grade = 8;
    else grade = 7;
    Console.WriteLine($"Grade Point: {grade}");
```

Q) Why validate before assigning to byte?
A) byte supports values only from 0–255. Direct casting without validation could cause overflow. Using conditional logic ensures only valid grade values are assigned, preventing runtime errors.

```
}

// Exercise 8: Mobile Data Usage Tracker
static void Exercise8()
```

```
{
    Console.Write("Enter data usage in bytes: ");
    long bytesUsed = long.Parse(Console.ReadLine());

    double mb = bytesUsed / (1024.0 * 1024);
    double gb = bytesUsed / (1024.0 * 1024 * 1024);
    int roundedGB = (int)Math.Round(gb);

    Console.WriteLine($"Usage: {mb:F2} MB");
    Console.WriteLine($"Rounded Usage: {roundedGB} GB");
```

Q) Why round instead of truncate data usage?

A) Rounding provides a user-friendly and realistic view of consumption. Truncation may under-report usage, which can mislead users about actual data consumption.

```
}

// Exercise 9: Warehouse Inventory Capacity
static void Exercise9()
{
    Console.Write("Enter current item count: ");
    int items = int.Parse(Console.ReadLine());
    Console.Write("Enter maximum capacity: ");
    ushort capacity = ushort.Parse(Console.ReadLine());
    bool isOverLimit = items > capacity;
    Console.WriteLine($"Over Capacity: {isOverLimit}");
```

Q) Why is comparing int with ushort risky?

A) int supports negative values, while ushort does not. Implicit conversion may lead to incorrect comparisons if negative values are involved. Explicit comparison ensures correctness and avoids logical errors.

```
    }

    // Exercise 10: Payroll Salary Computation
    static void Exercise10()
    {
        Console.Write("Enter basic salary: ");
        int basicSalary = int.Parse(Console.ReadLine());

        Console.Write("Enter allowance: ");
        double allowance = double.Parse(Console.ReadLine());

        Console.Write("Enter deduction: ");
        double deduction = double.Parse(Console.ReadLine());

        decimal netSalary =
            basicSalary +
            (decimal)allowance -
            (decimal)deduction;

        Console.WriteLine($"Net Salary: {netSalary}");
```

Q) Why not use double for net salary?
A) Salary calculations require exact precision. double may introduce rounding errors, while decimal ensures accuracy, making it ideal for payroll systems.

```
    }
}
```