Day 2 .NET EXERCISES

NAME: HARPUNEET KAUR
UID: 22BAI71102

EXERCISE 1

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace university_system
{
   internal class Class1
   {
      static void Main(string[] args)
      {
         Console.Write("Enter the no. of classes attended: ");
         int classesAttended = int.Parse(Console.ReadLine());
         Console.Write("Total no. of classes: ");
         int totalClassesHeld = int.Parse(Console.ReadLine());
         double attendancePercentage =
   (double)classesAttended / totalClassesHeld * 100;

         Console.WriteLine("Actual Percentage:"+attendancePercentage);


         //Truncate
         int displayPercentage = (int)attendancePercentage;
         Console.WriteLine("Attendance percentage: "+displayPercentage+"%");

         // Convert safely for display
         int eligibilityPercentage =
            (int)Math.Round(attendancePercentage, MidpointRounding.AwayFromZero);

         Console.WriteLine($"Attendance Roundof percentage: {eligibilityPercentage}%");

      }
   }
}
```

OUTPUT

```
Enter the no. of classes attended: 149
Total no. of classes: 200
Actual Percentage:74.5
Attendance percentage: 74%
Attendance Roundof percentage: 75%

C:\Users\HK\source\repos\Day2_ValueTypes_Conversions_Exercises\university_system\bin\Debug\net8.0\university_system.exe
(process 18596) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

149 attended out of 200 delivered → 74.5%

- Truncate → **74**
- Round → **75**

EXERCISE 2
SOURCE CODE

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace university_system
{
    internal class Class2
    {
        static void Main(string[] args)
        {

            int s1 = 78, s2 = 85, s3 = 91;

            int total = s1 + s2 + s3;

            // Step 1: Accurate average
            double average = (double)total / 3;

            // Step 2: Display with precision
            double displayAverage =
                Math.Round(average, 2, MidpointRounding.AwayFromZero);

            // Step 3: Scholarship conversion
```

```
        int scholarshipAverage =
            (int)Math.Round(displayAverage, MidpointRounding.AwayFromZero);

        Console.WriteLine($"Average: {displayAverage}");
        Console.WriteLine($"Scholarship Score: {scholarshipAverage}");

    }
}
```

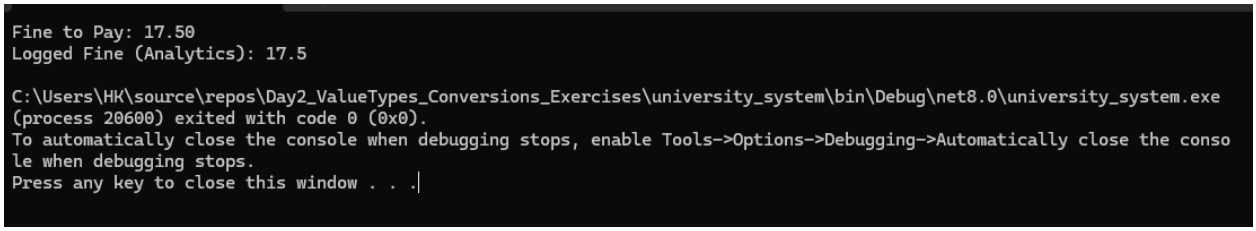
OUTPUT

EXERCISE 3
SOURCE CODE

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace university_system
{
    internal class Class3
    {
        static void Main(string[] args)
        {
            decimal finePerDay = 2.50m;
            int daysOverdue = 7;

            // Calculate total fine
            decimal totalFine = finePerDay * daysOverdue;

            // Display fine (rounded to 2 decimal places)
            decimal displayFine =
                Math.Round(totalFine, 2, MidpointRounding.AwayFromZero);

            // Log fine for analytics
            double analyticsFine = (double)displayFine;
```

```
        Console.WriteLine($"Fine to Pay: ₹{displayFine}");
        Console.WriteLine($"Logged Fine (Analytics): {analyticsFine}");

    }
  }
}
```

OUTPUT



Short Explanation

- **`decimal` is used for fine amounts to ensure accurate financial calculations without rounding errors.**
- **`int` is used for days overdue because days are whole numbers.**
- **The total fine is displayed as `decimal` to maintain currency precision.**
- **The fine is logged as `double` for analytics due to better performance and tool compatibility.**

Conversions:

- **`int` to `decimal` occurs implicitly during calculation and is safe.**
- **`decimal` to `double` requires explicit casting and may cause minor precision loss, which is acceptable for analytics.**

EXERCISE 4
SOURCE CODE

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace university_system
{
   internal class Class4
```

```
    {
        static void Main(string[] args)
        {
            decimal accountBalance = 100000.00m;

            // Interest rate from external API (float)
            float interestRate = 7.5f;   // annual percentage rate

            // Explicit conversion of float to decimal
            decimal rateDecimal = (decimal)interestRate;

            // Monthly interest calculation
            decimal monthlyInterest =
                accountBalance * rateDecimal / 100 / 12;

            // Update balance
            accountBalance += monthlyInterest;

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

        }
    }
}
```

OUTPUT

```
Updated Balance: 100625.000

C:\Users\HK\source\repos\Day2_ValueTypes_Conversions_Exercises\university_system\bin\Debug\net8.0\university_system.exe
(process 26344) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

Why Types Are Used & Conversions

- **`decimal` is used for account balance to ensure accurate financial calculations.**
- **`float` is used for interest rate because it comes from an external API and is suitable for approximate values.**
- **`float` cannot be implicitly converted to `decimal` because it may cause precision loss.**
- **Therefore, an explicit cast is required to safely convert `float` to `decimal`.**
- **All monetary calculations are performed in `decimal` to avoid rounding errors.**

Why Implicit Conversion May Fail

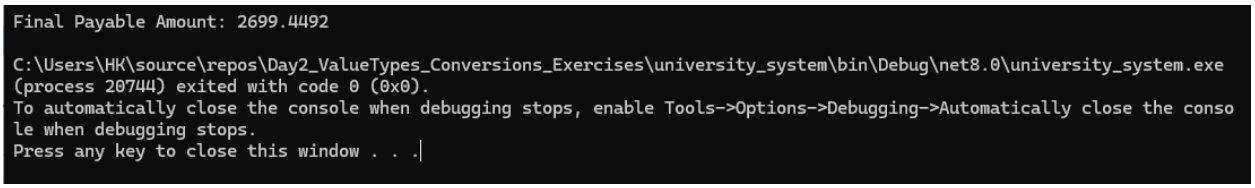
- **`float` is a binary floating-point type and may not represent decimal values exactly.**
- **`decimal` is a high-precision base-10 type.**
- **Implicit conversion is disallowed by C# to prevent unintended financial inaccuracies.**

EXERCISE 5
SOURCE CODE

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace university_system
{
   internal class Class5
   {
      static void Main(string[] args)
      {
         // Cart total accumulated from item prices
         double cartTotal = 1999.99 + 499.50;

         // Tax and discount rules (financial logic)
         decimal taxRate = 18.0m;      // 18%
         decimal discountRate = 10.0m; // 10%

         // Convert cart total to decimal before financial calculations
         decimal cartTotalDecimal = (decimal)cartTotal;

         // Apply tax and discount
         decimal taxAmount = cartTotalDecimal * taxRate / 100;
         decimal discountAmount = cartTotalDecimal * discountRate / 100;

         // Final payable amount
         decimal finalAmount = cartTotalDecimal + taxAmount - discountAmount;

         Console.WriteLine($"Final Payable Amount: {finalAmount}");

      }
```

```
    }
}
```

OUTPUT



Explanation: Conversion Strategy

- **`double`** is used for cart accumulation because it is fast and commonly used for intermediate calculations.
- **`decimal`** is required for tax, discount, and final payable amount to ensure financial accuracy.
- The cart total is explicitly converted from **`double`** to **`decimal`** before applying tax and discount.
- All monetary rules are applied only after conversion to decimal.

Precision Risks

- **`double`** uses binary floating-point representation and may introduce small rounding errors.
- Converting an imprecise **`double`** to **`decimal`** can carry forward these inaccuracies.
- Therefore, conversion is done once, early, and all financial calculations are handled in **`decimal`**.
- Using **`double`** throughout could result in incorrect billing amounts.

EXERCISE 6
SOURCE CODE

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace university_system
{
    internal class Class6
```

```
  {
    static void Main(string[] args)
    {
        // Temperature reading from sensor (e.g., tenths of °C)
        short sensorReading = 325; // represents 32.5°C

        // Convert to Celsius (double for precision)
        double temperatureCelsius = sensorReading / 10.0;

        // Store multiple readings (example)
        double[] dailyReadings = { 32.5, 33.1, 31.8 };

        // Calculate daily average
        double dailyAverage =
            dailyReadings.Average();

        // Convert average for dashboard display
        int displayTemperature =
            (int)Math.Round(dailyAverage, MidpointRounding.AwayFromZero);

        Console.WriteLine($"Average Temperature: {displayTemperature}°C");

    }
  }
}
```

OUTPUT

```
Average Temperature: 32°C

C:\Users\HK\source\repos\Day2_ValueTypes_Conversions_Exercises\university_system\bin\Debug\net8.0\university_system.exe
(process 23856) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

Overflow Concerns

- **Arithmetic on `short` values is implicitly promoted to `int` in C#.**
- **This prevents overflow during calculations.**
- **Overflow risk exists only if sensor values exceed the `short` range, which must be validated at input.**

EXERCISE 7
SOURCE CODE

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace university_system
{
    internal class Class7
    {
        static void Main(string[] args)
        {
            // Final score after evaluation
            double finalScore = 87.6;

            // Validate score range
            if (finalScore < 0 || finalScore > 100)
            {
                throw new ArgumentOutOfRangeException(nameof(finalScore),
                    "Score must be between 0 and 100.");
            }

            // Convert score to integer percentage
            int roundedScore =
                (int)Math.Round(finalScore, MidpointRounding.AwayFromZero);

            // Map score to grade (stored as byte)
            byte grade;

            if (roundedScore >= 90)
                grade = 10;       // A+
            else if (roundedScore >= 80)
                grade = 9;        // A
            else if (roundedScore >= 70)
                grade = 8;        // B
            else if (roundedScore >= 60)
                grade = 7;        // C
            else if (roundedScore >= 50)
                grade = 6;        // D
            else
                grade = 5;        // Fail

            Console.WriteLine($"Score: {finalScore}");
            Console.WriteLine($"Grade Code: {grade}");
```

```
    }
  }
}
```

OUTPUT

```
Score: 87.6
Grade Code: 9

C:\Users\HK\source\repos\Day2_ValueTypes_Conversions_Exercises\university_system\bin\Debug\net8.0\university_system.exe
(process 24492) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

Validation Choices

- **Score is validated to be within 0–100 before conversion.**
- **Prevents invalid values that could cause incorrect grading or overflow.**

Casting Choices

- **`double` → `int` conversion uses rounding, not truncation.**
  - **Prevents unfair grade reduction.**
- **`int` → `byte` assignment is safe because grade values are controlled and within range.**

EXERCISE 8
SOURCE CODE

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace university_system
{
   internal class Class8
   {
      static void Main(string[] args)
      {
         // Data usage tracked in bytes
         long usageBytes = 5_368_709_120; // example value

         // Implicit conversion: long → double
```

```
        double usageMB = usageBytes / (1024.0 * 1024.0);
        double usageGB = usageBytes / (1024.0 * 1024.0 * 1024.0);

        // Monthly summary (rounded to nearest integer)
        int roundedMB =
            (int)Math.Round(usageMB, MidpointRounding.AwayFromZero);

        int roundedGB =
            (int)Math.Round(usageGB, MidpointRounding.AwayFromZero);

        Console.WriteLine($"Usage: {usageMB:F2} MB");
        Console.WriteLine($"Usage: {usageGB:F2} GB");
        Console.WriteLine($"Monthly Summary: {roundedGB} GB");

    }
  }
}
```

OUTPUT

```
Usage: 5120.00 MB
Usage: 5.00 GB
Monthly Summary: 5 GB

C:\Users\HK\source\repos\Day2_ValueTypes_Conversions_Exercises\university_system\bin\Debug\net8.0\university_system.exe
(process 13308) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

double usageMB = usageBytes / (1024.0 * 1024.0);
long is implicitly converted to double.
Safe because double can represent large integers approximately.
Enables floating-point division.

Math.Round(value, MidpointRounding.AwayFromZero);
.5 values round up.
Rounding is preferred over truncation for monthly summaries to avoid misleading reports.

EXERCISE 9
SOURCE CODE

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace university_system
{
    internal class Class9
    {
        static void Main(string[] args)
        {
            // Current inventory count (can be zero or positive)
            int itemCount = 450;

            // Maximum warehouse capacity (always non-negative)
            ushort maxCapacity = 500;

            // Convert capacity for safe comparison
            int capacityForComparison = maxCapacity;

            // Capacity check
            if (itemCount > capacityForComparison)
            {
                Console.WriteLine("Warning: Inventory exceeds maximum capacity!");
            }
            else
            {
                Console.WriteLine("Inventory is within capacity.");
            }

            // Reporting
            Console.WriteLine($"Items Stored: {itemCount}");
            Console.WriteLine($"Maximum Capacity: {capacityForComparison}");

        }
    }
}
```

OUTPUT

```
Inventory is within capacity.
Items Stored: 450
Maximum Capacity: 500

C:\Users\HK\source\repos\Day2_ValueTypes_Conversions_Exercises\university_system\bin\Debug\net8.0\university_system.exe
(process 16360) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

- **Item count is stored as a signed integer for flexibility, while maximum capacity is stored as an unsigned type to prevent negative values**

- **capacity is safely converted to int for comparison to avoid signed–unsigned conversion risks such as overflow and wrap-around errors.**

EXERCISE 10
SOURCE CODE

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Exercises
{
    internal class Class10
    {
        static void Main(string[] args)
        {
            // Basic salary (fixed whole amount)
            int basicSalary = 50000;

            // Variable components
            double allowances = 12450.75;
            double deductions = 3250.25;

            // Convert to decimal for accurate payroll calculation
            decimal basicDecimal = basicSalary;
            decimal allowancesDecimal = (decimal)allowances;
            decimal deductionsDecimal = (decimal)deductions;
            // Net salary computation
            decimal netSalary = basicDecimal + allowancesDecimal - deductionsDecimal;

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

        }
    }
}
```

OUTPUT

```
Net Salary: 59200.50

C:\Users\HK\source\repos\Day2_ValueTypes_Conversions_Exercises\university_system\bin\Debug\net8.0\Exercises.exe (process
 6172) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

(Justification of Type Choices & Conversions)

- **`int` for basic salary**
  Basic salary is a fixed whole number, so `int` is sufficient and efficient.
- **`double` for allowances and deductions**
  These values may come from external systems or calculations and can contain fractional values.
- **`decimal` for net salary**
  Payroll calculations require high precision; `decimal` avoids floating-point rounding errors.