Lecture 03 - Practice Tasks

Arrays, Classes & Strings in C#

Created by: ITI

Tasks Overview

#	Task Name	Topic
1	Array Rotation	Single Dimension Arrays
2	Spiral Matrix	Two Dimension Arrays
3	Pascal's Triangle	Jagged Arrays
4	Sorting Algorithms	Array Methods
5	var Limitations	Implicitly Typed Variables
6	Bank Account System	Classes & Objects
7	Array Utility Class	Static Classes
8	Word Frequency Counter	String Methods

Task 1: Array Rotation

Description

Write a program that rotates an array K positions to the right without using a second array (in-place rotation).

The last K elements should move to the beginning of the array, and all other elements shift to the right.

Example

```
Input Array: [1, 2, 3, 4, 5]
Rotation Count (K): 2

Output: [4, 5, 1, 2, 3]
```

Illustration

```
BEFORE ROTATION:

┌─────┬─────┬─────┬─────┬─────┐
│  1  │  2  │  3  │  4  │  5  │
└─────┴─────┴─────┴─────┴─────┘
  [0]   [1]   [2]   [3]   [4]


K = 2  →  Last 2 elements move to front


AFTER ROTATION:

┌─────┬─────┬─────┬─────┬─────┐
│  4  │  5  │  1  │  2  │  3  │
└─────┴─────┴─────┴─────┴─────┘
  [0]   [1]   [2]   [3]   [4]
   ↑     ↑
   └─────┘── These came from the end!
```

Another Example

```
Array: [A, B, C, D, E, F]    K = 3

Result: [D, E, F, A, B, C]

The last 3 elements (D, E, F) moved to the front.
```

Task 2: Spiral Matrix

Description

Fill an N×N matrix with numbers from 1 to N^2 in a clockwise spiral pattern.

Start from the top-left corner and move: Right → Down → Left → Up → and repeat going inward.

Example

```
Input: N = 4

Output:
```

1	2	3	4
12	13	14	5
11	16	15	6
10	9	8	7

Illustration

```
THE SPIRAL PATH:

Start here
    ↓
    1  →  2  →  3  →  4
                      ↓
   12 → 13 → 14       5
    ↑         ↓       ↓
   11    16 ← 15      6
    ↑                 ↓
   10  ←  9  ←  8  ←  7


DIRECTION ORDER:
  1. Go RIGHT  →  →  →
  2. Go DOWN   ↓
  3. Go LEFT   ←  ←  ←
  4. Go UP     ↑
  5. Repeat (spiral inward)


3x3 EXAMPLE:
```

```
┌───┬───┬───┐
│ 1 │ 2 │ 3 │  →  Fill top row (1,2,3)
├───┼───┼───┤
│ 8 │ 9 │ 4 │  ↓  Fill right column (4)
├───┼───┼───┤     Fill bottom row (5,6,7)
│ 7 │ 6 │ 5 │  ←  Fill left column (8)
└───┴───┴───┘     Fill center (9)
```

Task 3: Pascal's Triangle

Description

Generate Pascal's Triangle with N rows using a jagged array.

Each number is the sum of the two numbers directly above it. The first and last number in each row is always 1.

Example

```
Input: N = 6

Output:
          1
         1 1
        1 2 1
       1 3 3 1
      1 4 6 4 1
     1 5 10 10 5 1
```

Illustration

```
HOW PASCAL'S TRIANGLE WORKS:

Row 0:            1                    → 1 element
Row 1:          1 1                    → 2 elements
Row 2:        1 2 1                    → 3 elements
Row 3:      1 3 3 1                    → 4 elements
Row 4:    1 4 6 4 1                    → 5 elements
```

```
THE ADDITION RULE:

Each number = sum of two numbers above it

           1
          / \
         1   1
        / \ / \
       1   2   1         ← 1+1 = 2
      / \ / \ / \
     1   3   3   1       ← 1+2=3, 2+1=3
    / \ / \ / \ / \
   1   4   6   4   1     ← 1+3=4, 3+3=6, 3+1=4


JAGGED ARRAY STRUCTURE:
Row 0: [1]              → array of 1 element
Row 1: [1, 1]           → array of 2 elements
Row 2: [1, 2, 1]        → array of 3 elements
Row 3: [1, 3, 3, 1]     → array of 4 elements

Each row is a separate array with different length!
```

Task 4: Sorting Algorithms

Description

Implement two sorting algorithms from scratch:

1. **Bubble Sort** - Compare adjacent elements and swap if needed
2. **Selection Sort** - Find minimum and place it in correct position

Compare your results with the built-in Array.Sort() to verify correctness.

Example

```
Input:  [64, 34, 25, 12, 22, 11, 90]

Output: [11, 12, 22, 25, 34, 64, 90]
```

Illustration - Bubble Sort

```
BUBBLE SORT: Compare neighbors, swap if out of order
             Largest "bubbles up" to the end

Original: [5] [3] [8] [1]

Pass 1:
  [5] [3] [8] [1]    Compare 5,3 → Swap!
   ↔   ↔
  [3] [5] [8] [1]    Compare 5,8 → OK
       ↔   ↔
  [3] [5] [8] [1]    Compare 8,1 → Swap!
            ↔   ↔
  [3] [5] [1] [8]    ← 8 is now at the end (sorted)

Pass 2:
  [3] [5] [1] [8]    Compare 3,5 → OK
  [3] [5] [1] [8]    Compare 5,1 → Swap!
  [3] [1] [5] [8]    ← 5 is in place

Pass 3:
  [3] [1] [5] [8]    Compare 3,1 → Swap!
  [1] [3] [5] [8]    ← SORTED!
```

Illustration - Selection Sort

```
SELECTION SORT: Find the smallest, put it first
                Then find next smallest, and so on

Original: [5] [3] [8] [1]

Step 1: Find minimum in whole array
  [5] [3] [8] [1]
              ↑
           min = 1
  Swap with first position:
  [1] [3] [8] [5]    ← 1 is now in correct place

Step 2: Find minimum in remaining [3] [8] [5]
  [1] [3] [8] [5]
```

```
        ↑
    min = 3
  Already in correct position!
  [1] [3] [8] [5]    ← 3 is now in correct place

Step 3: Find minimum in remaining [8] [5]
  [1] [3] [8] [5]
                ↑
            min = 5
  Swap with position 2:
  [1] [3] [5] [8]    ← SORTED!
```

Task 5: var Limitations

Description

Create a program that demonstrates all 6 limitations of the var keyword in C#.

Show what works and what causes errors.

Example

```
WHAT WORKS:

var number = 10;              ✓  Type: int
var name = "Ahmed";           ✓  Type: string
var prices = new[] {1,2,3};   ✓  Type: int[]


WHAT DOES NOT WORK:

var x;                        ✗  Error: Must initialize!
var y = null;                 ✗  Error: Cannot be null!
var a = 1, b = 2;             ✗  Error: One at a time!
```

Illustration - The 6 Limitations

```
╔═══════════════════════════════════════════════════════════════════════╗
║                      var KEYWORD LIMITATIONS                          ║
```

```
╔═══════════════════════════════════════════════════════════════╗
║                                                               ║
║  1. MUST INITIALIZE IMMEDIATELY                               ║
║     ✗ var x;          // Error!                               ║
║     ✓ var x = 10;     // OK                                   ║
║                                                               ║
║  2. CANNOT BE NULL                                            ║
║     ✗ var y = null;  // Error! What type is null?             ║
║     ✓ string y = null; // OK with explicit type               ║
║                                                               ║
║  3. CANNOT USE AS CLASS FIELD                                 ║
║     class MyClass {                                           ║
║       ✗ var field = 10;     // Error!                         ║
║       ✓ int field = 10;     // OK                             ║
║     }                                                         ║
║                                                               ║
║  4. CANNOT USE IN OWN INITIALIZATION                          ║
║     ✗ var i = i + 1;  // Error! i doesn't exist yet           ║
║     ✓ int i = 0;                                              ║
║       i = i + 1;      // OK                                   ║
║                                                               ║
║  5. CANNOT DECLARE MULTIPLE                                   ║
║     ✗ var a = 1, b = 2;   // Error!                           ║
║     ✓ var a = 1;                                              ║
║       var b = 2;          // OK                               ║
║                                                               ║
║  6. CANNOT USE AS RETURN TYPE                                 ║
║     ✗ static var GetValue() { }   // Error!                   ║
║     ✓ static int GetValue() { }   // OK                       ║
║                                                               ║
╚═══════════════════════════════════════════════════════════════╝
```

Task 6: Bank Account System

Description

Create a BankAccount class with:

- Private fields: accountNumber, balance, ownerName
- Methods: Deposit, Withdraw, Transfer, GetBalance, DisplayInfo

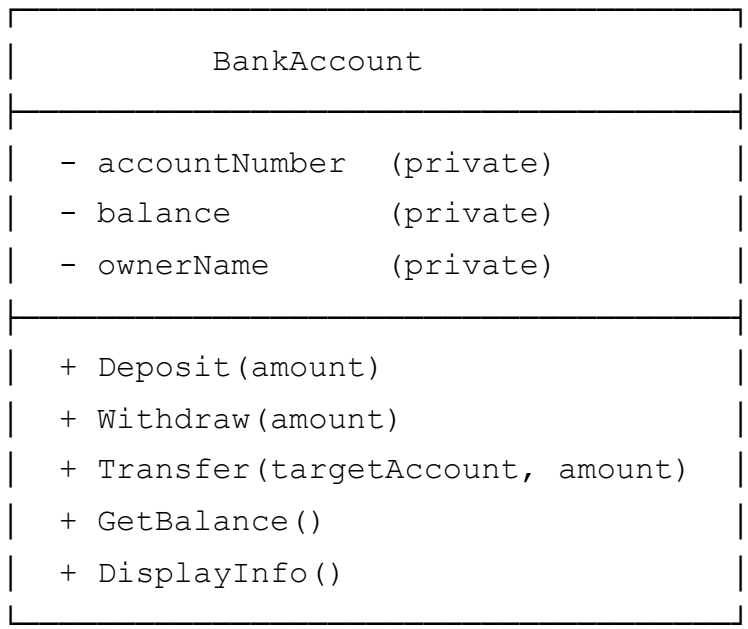
Demonstrate how reference types work when passing objects to methods.

Example

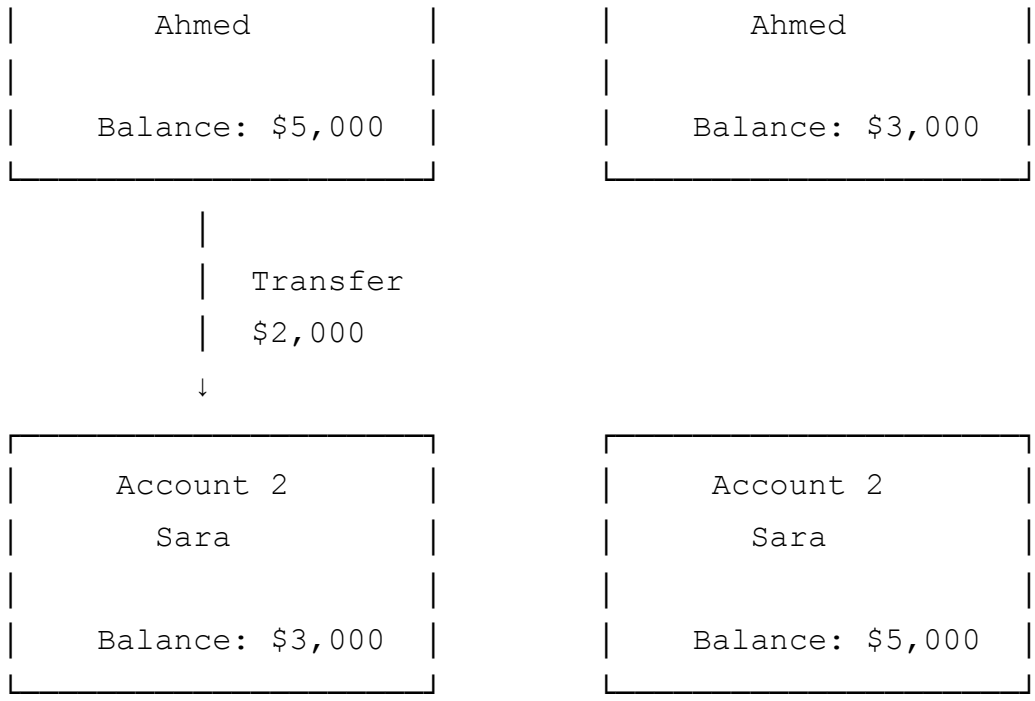
```
Create two accounts:
  Account1: Ahmed, Balance = $5,000
  Account2: Sara,  Balance = $3,000

Operations:
  Ahmed deposits $1,000     → Ahmed: $6,000
  Ahmed withdraws $500      → Ahmed: $5,500
  Ahmed transfers $2,000 to Sara

Result:
  Account1: Ahmed, Balance = $3,500
  Account2: Sara,  Balance = $5,000
```

Illustration

BANK ACCOUNT CLASS STRUCTURE:

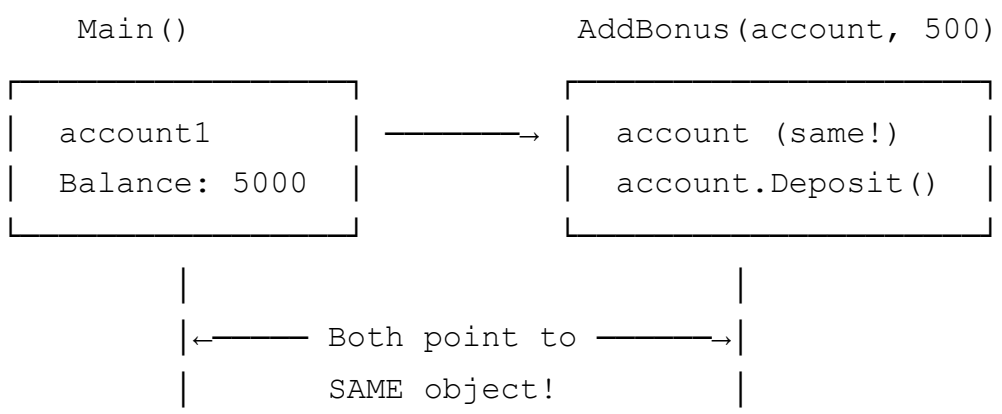


TRANSFER OPERATION:



REFERENCE TYPE BEHAVIOR:

When you pass an account to a method,
the method can modify the original account!



Task 7: Array Utility Class

Description

Create a **static** class called ArrayUtils with these static methods:

- Reverse(arr) - Reverse array in place
- FindMax(arr) - Return maximum value
- FindMin(arr) - Return minimum value
- IsSorted(arr) - Check if sorted

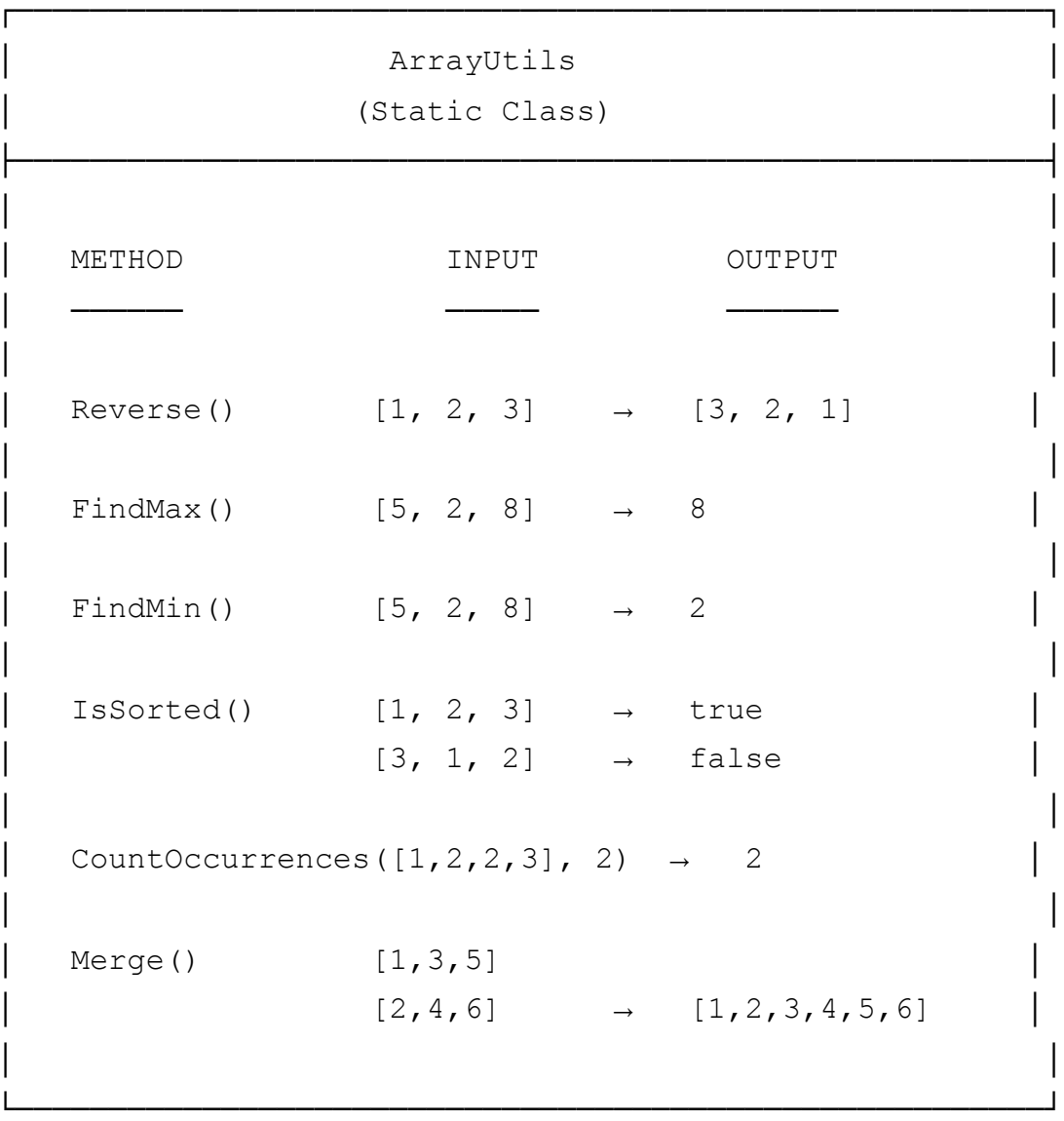
- CountOccurrences(arr, value) - Count how many times value appears
- Merge(arr1, arr2) - Merge two sorted arrays

Example

```
Array: [5, 2, 8, 1, 9]

ArrayUtils.FindMax(arr)   → 9
ArrayUtils.FindMin(arr)   → 1
ArrayUtils.IsSorted(arr)  → false
ArrayUtils.Reverse(arr)   → [9, 1, 8, 2, 5]

ArrayUtils.Merge([1,3,5], [2,4,6]) → [1,2,3,4,5,6]
```

Illustration

```
STATIC CLASS - NO OBJECT CREATION NEEDED!
```



```
HOW TO USE:

  ✗ WRONG:
    ArrayUtils utils = new ArrayUtils();  // Cannot create object!
    utils.FindMax(arr);

  ✓ CORRECT:
    ArrayUtils.FindMax(arr);              // Call directly on class!
    ArrayUtils.Reverse(arr);


MERGE OPERATION VISUALIZED:

  Array 1: [1, 3, 5, 7]      (sorted)
  Array 2: [2, 4, 6]         (sorted)
           ↓
  Compare and pick smaller each time:
           ↓
  Result:  [1, 2, 3, 4, 5, 6, 7]  (merged & sorted)
```

Task 8: Word Frequency Counter

Description

Write a program that:

1. Takes a sentence as input
2. Counts how many times each word appears (case-insensitive)
3. Displays results sorted by frequency (highest first)

Use string methods: ToLower(), Compare(), Concat()

Example

```
Input: "The cat and the dog and the bird"

Output:
  the  - 3
```

```
and  - 2
cat  - 1
dog  - 1
bird - 1
```

Illustration

STEP-BY-STEP PROCESS:

Step 1: INPUT

```
┌──────────────────────────────────────────────────────┐
│  "The cat and the dog and the bird"                  │
└──────────────────────────────────────────────────────┘
```

Step 2: CONVERT TO LOWERCASE (using ToLower())

```
┌──────────────────────────────────────────────────────┐
│  "the cat and the dog and the bird"                  │
└──────────────────────────────────────────────────────┘
```

Step 3: SPLIT INTO WORDS

```
┌─────┬─────┬─────┬─────┬─────┬─────┬─────┬──────┐
│ the │ cat │ and │ the │ dog │ and │ the │ bird │
└─────┴─────┴─────┴─────┴─────┴─────┴─────┴──────┘
```

Step 4: COUNT EACH WORD

```
┌──────────────────────────────────┐
│  Word      │  Count              │
├────────────┼─────────────────────┤
│  the       │  |||     = 3        │
│  cat       │  |       = 1        │
│  and       │  ||      = 2        │
│  dog       │  |       = 1        │
│  bird      │  |       = 1        │
└────────────┴─────────────────────┘
```

Step 5: SORT BY FREQUENCY (highest to lowest)

```
┌──────────────────────────────────┐
│  Word       │  Frequency         │
├─────────────┼────────────────────┤
│  the        │     3              │
│  and        │     2              │
│  cat        │     1              │
│  dog        │     1              │
│  bird       │     1              │
└─────────────┴────────────────────┘
```

```
STRING METHODS USED:

  ToLower()    →  "Hello" becomes "hello"
  Compare()    →  Compare two strings (0 if equal)
  Concat()     →  Join strings together
```

Good Luck!

Notes:

- Test your code with different inputs
- Make sure your code compiles without errors
- Add comments to explain your logic
- Check edge cases (empty array, single element, etc.)

Created by: ITI