C#


C# Data types
Value Types
Reference Types

1. Value types
    1. byte
    2. short
    3. int
    4. long
    5. float
    6. double
    7. decimal
    8. bool
    9. char

2. Reference Types

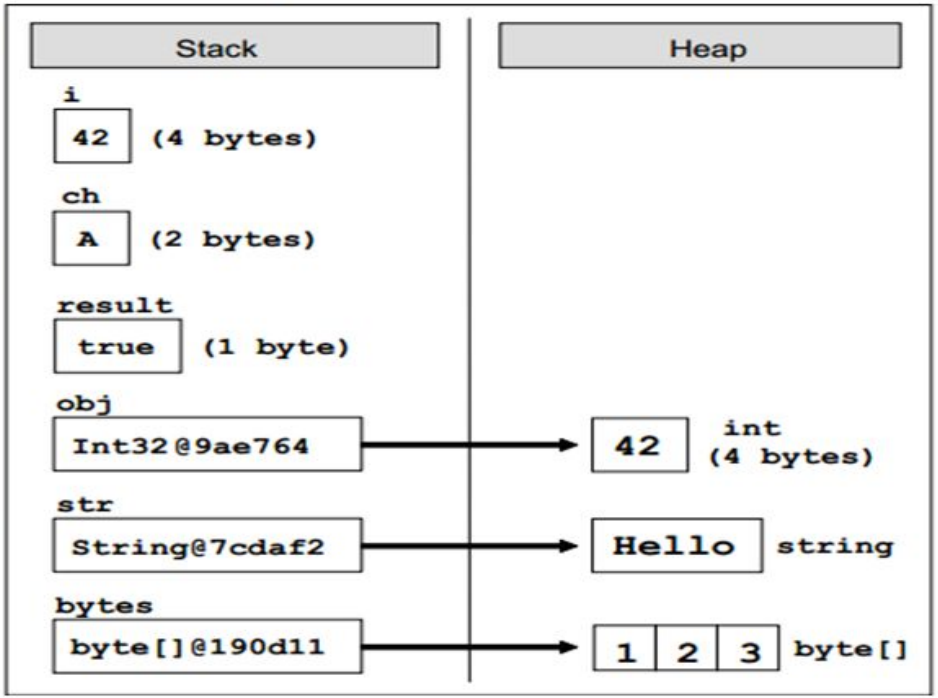
1. Strings
2. Objects
3. Classes
4. Interfaces

VALUE TYPES

DATA TYPES	SIZE	VALUES
sbyte	8 bit	-128 to 127
byte	8 bit	0 to 255
short	16 bit	-32,768 to 32,767
ushort	16 bit	0 to 65,535
int	32 bit	-2,147,483,648 to 2,147,483,647
uint	32 bit	0 to 4,294,967,295
long	64 bit	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
ulong	64 bit	0 to 18,446,744,073,709,551,615
char	16 bit	0 to 65535
float	32 bit	-1.5 x 1045 to 3.4 x 1038
double	64 bit	-5 x 10324 to 1.7 x 10308
decimal	128 bit	-1028 to 7.9 x 1028
bool	—	True or false

- **Value types** are stored directly in the **stack**, meaning their actual data is kept there.
- **Reference types** store their references (pointers) in the **stack**, but their actual objects are allocated in the **heap**.

```
int i = 42;
char ch = 'A';
bool result = true;
object obj = 42;
string str = "Hello";
byte[] bytes ={1,2,3};
```



Loop in C#

1. for loop
2. while
3. do while
4. foreach

```
1  //For Loop Example:
2  // Print numbers from 1 to 10
3  for (int i = 1; i <= 10; i++)
4  {
5      Console.WriteLine("Number: " + i);
6  }
7
```


//While Loop Example:
// Keep asking for input until "exit" is entered
string input = "";
while (input != "exit")
{
Console.WriteLine("Enter 'exit' to quit:");
input = Console.ReadLine();
}

```
1 //Do-While Loop Example:
2 // Execute the code block at least once, then repeat as long as
  the condition is true
3 int number;
4 do
5 {
6     Console.WriteLine("Enter a number greater than 10:");
7     number = int.Parse(Console.ReadLine());
8 } while (number <= 10);
```

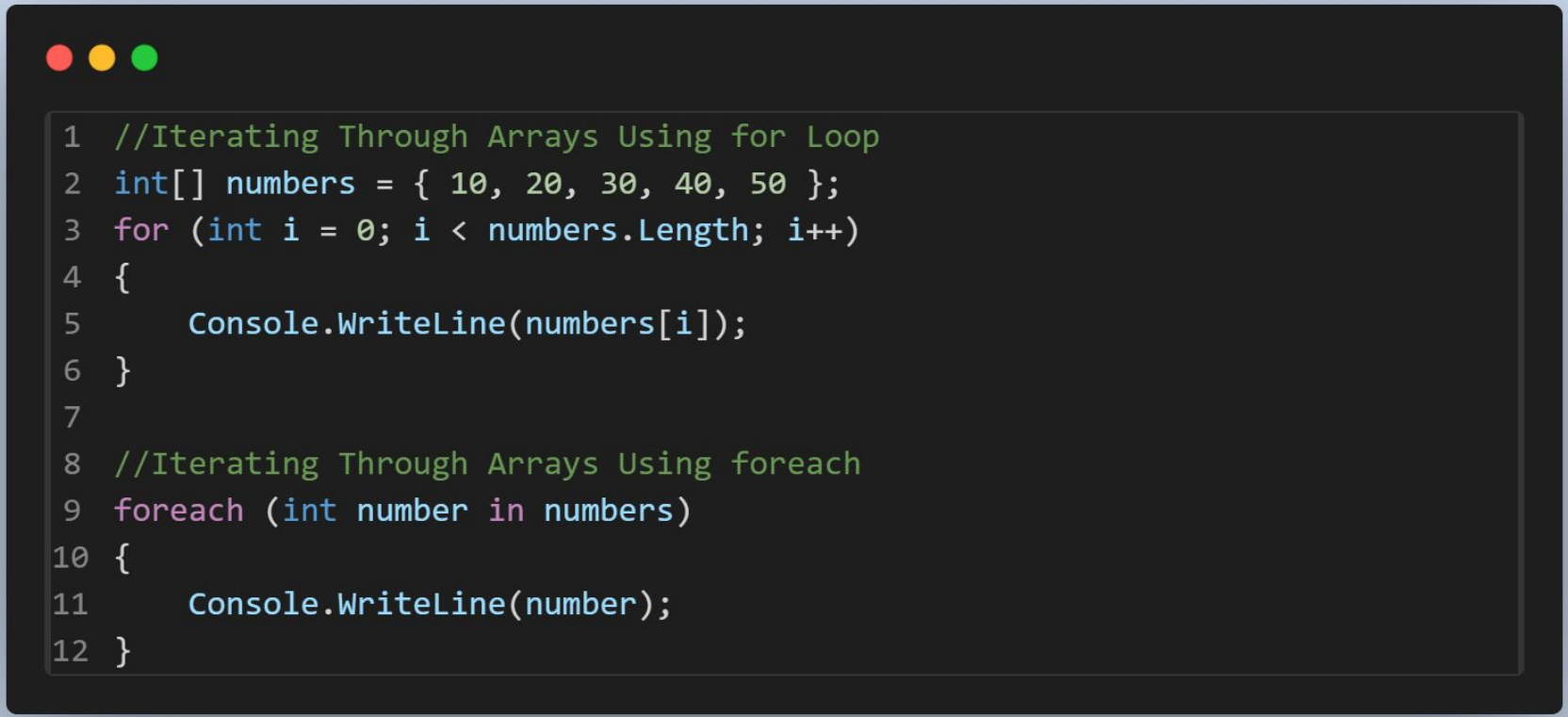
```
Foreach Loop Example:
// Iterate through the elements of an array
string[] colors = { "Red", "Green", "Blue" };
foreach (string color in colors)
{
    Console.WriteLine("Color: " + color);
}
```

Array in C#

An **array** is a collection of elements of the same type. In C#, there 3 types of arrays

1. Single-Dimensional Array
2. Multi-Dimensional Array
3. Jagged Array (Array of Arrays)

```
1  //Declaring Single-Dimensional Arrays
2
3  // Creates an array of 5 integers
4  int[] numbers = new int[5];
5
6  // Creates an array of 3 strings
7  string[] names = new string[3];
8
9
10 //Initializing Arrays
11 int[] numbers = { 10, 20, 30, 40, 50 };
12 string[] names = { "Alice", "Bob", "Charlie" };
13
14 //Accessing Array Elements
15 int[] numbers = { 10, 20, 30, 40, 50 };
16 Console.WriteLine(numbers[0]); // Output: 10
17 Console.WriteLine(numbers[2]); // Output: 30
18
```


//Iterating Through Arrays Using for Loop
int[] numbers = { 10, 20, 30, 40, 50 };
for (int i = 0; i < numbers.Length; i++)
{
Console.WriteLine(numbers[i]);
}
//Iterating Through Arrays Using foreach
foreach (int number in numbers)
{
Console.WriteLine(number);
}

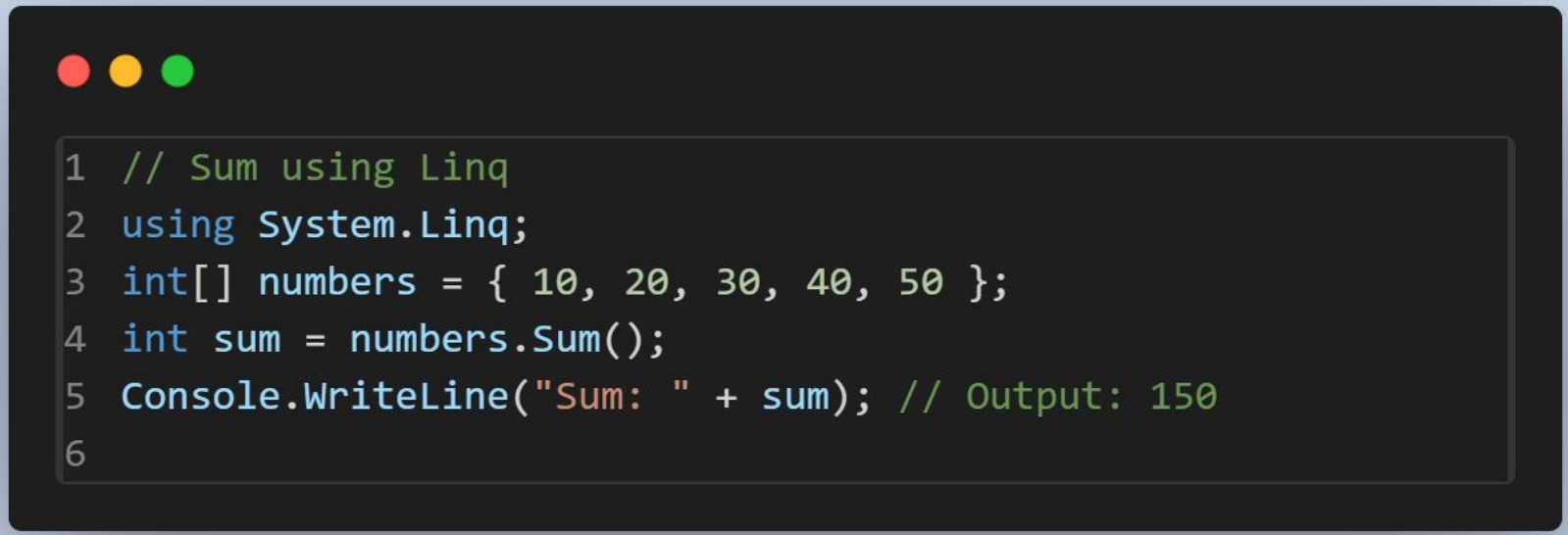
```
// Declare and initialize a multidimensional  array
int[,] matrix = {
    { 1, 2, 3 },
    { 4, 5, 6 },
    { 7, 8, 9 }
};

// Get the number of rows and columns
int rows = matrix.GetLength(0); // Number of rows
int cols = matrix.GetLength(1); // Number of columns

// Iterate through the array
for (int i = 0; i < rows; i++) // Loop through rows
{
    for (int j = 0; j < cols; j++) // Loop through columns
    {
        Console.Write(matrix[i, j] + " "); // Access element at [i, j]
    }
    Console.WriteLine(); // Move to the next line after each row
}

// output
1 2 3
4 5 6
7 8 9
```

Jagged array

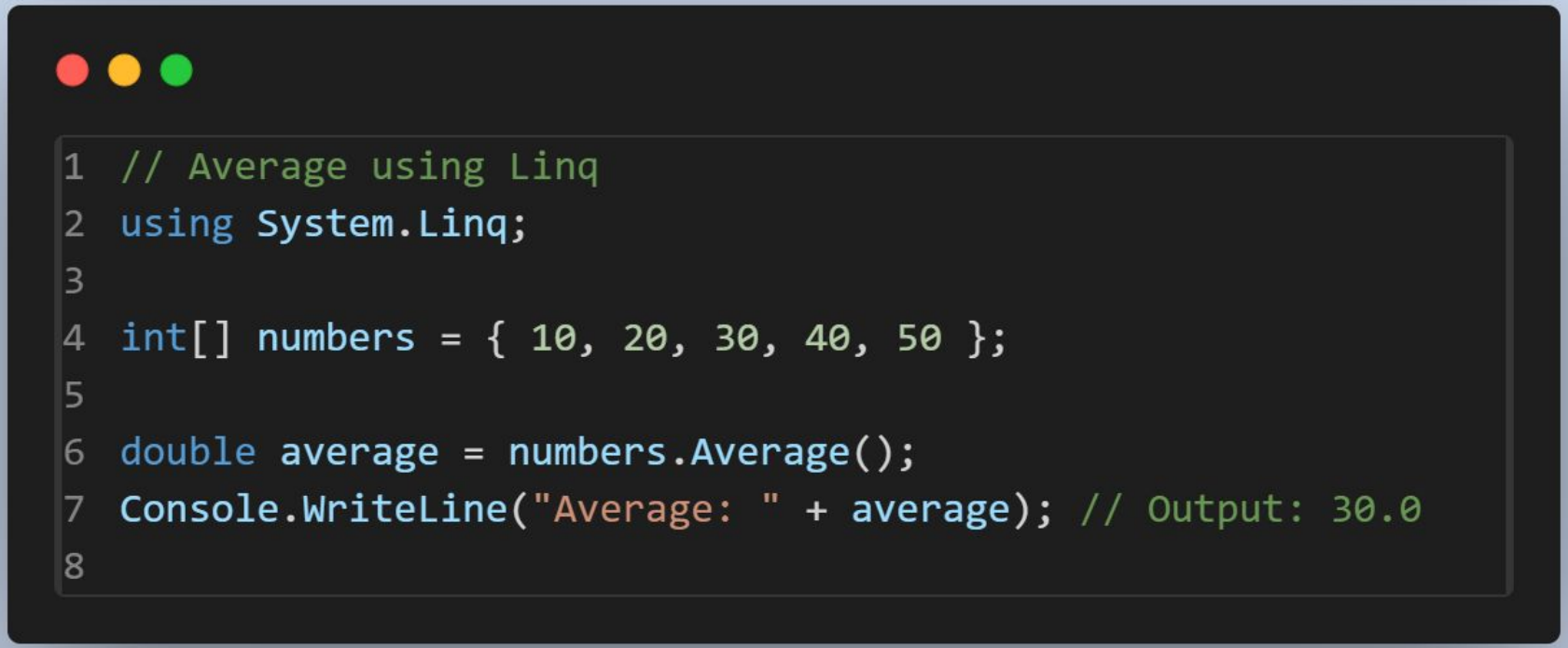
- A jagged array in C# is an array of arrays , where each element of the main array is itself an array.
- Unlike multidimensional arrays (e.g., int[,]), jagged arrays allow each sub-array to have a different length .
- This flexibility makes them useful for scenarios where rows (or columns) have varying sizes.

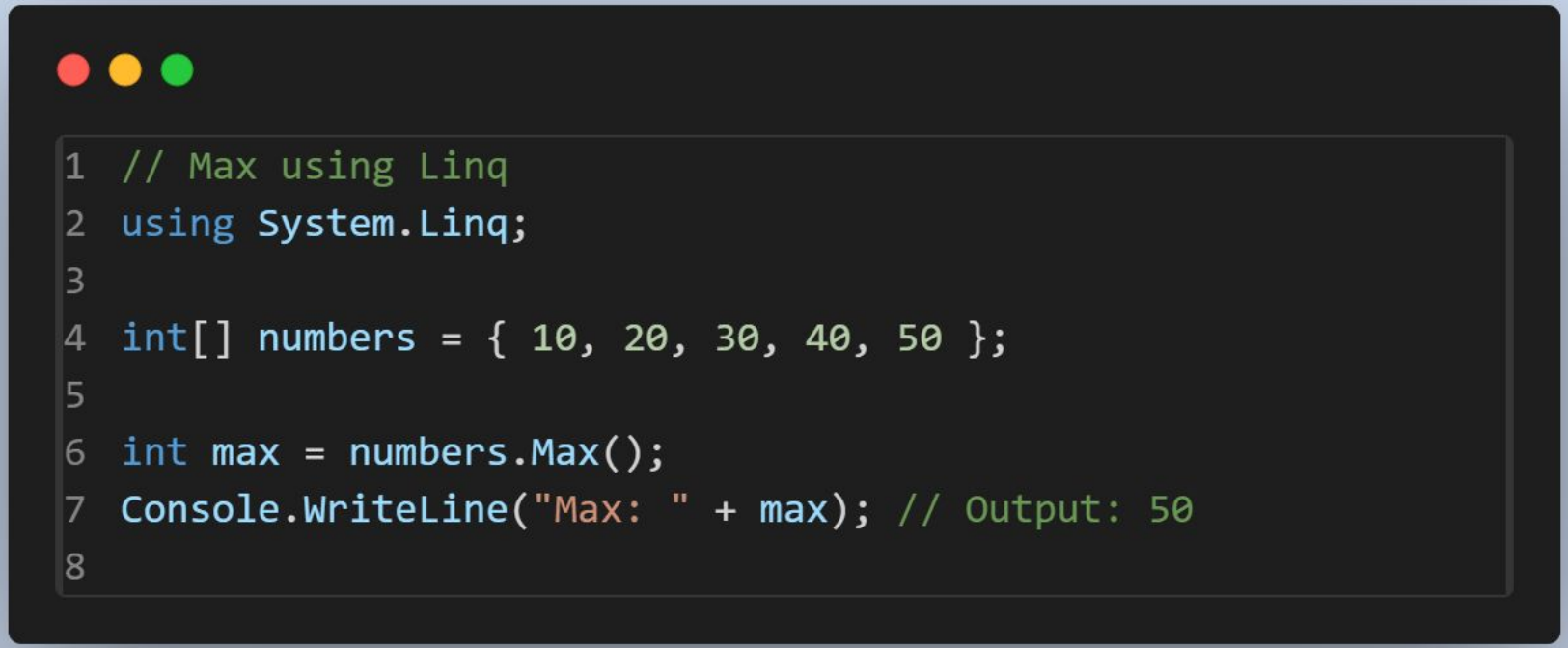
```
//Jagged Array Example:
int[][] jaggedArray = new int[][]
{
    new int[] { 1, 2, 3 },
    new int[] { 4, 5 },
    new int[] { 6 }
};

// Outer loop iterates through rows
for (int i = 0; i < jaggedArray.Length; i++)
{

// Inner loop iterates through elements in the current row
    for (int j = 0; j < jaggedArray[i].Length; j++)
    {
        Console.Write(jaggedArray[i][j] + " ");
// Print each element
    }
    Console.WriteLine();
// Move to the next line after each row
}
//Output
1 2 3
4 5
6
```

Some Common Array Operation



```
// Sum using Linq
using System.Linq;
int[] numbers = { 10, 20, 30, 40, 50 };
int sum = numbers.Sum();
Console.WriteLine("Sum: " + sum); // Output: 150

```


// Average using Linq
using System.Linq;
int[] numbers = { 10, 20, 30, 40, 50 };
double average = numbers.Average();
Console.WriteLine("Average: " + average); // Output: 30.0


// Max using Linq
using System.Linq;
int[] numbers = { 10, 20, 30, 40, 50 };
int max = numbers.Max();
Console.WriteLine("Max: " + max); // Output: 50


// Min using Linq
using System.Linq;
int[] numbers = { 10, 20, 30, 40, 50 };
int min = numbers.Min();
Console.WriteLine("Min: " + min); // Output: 10

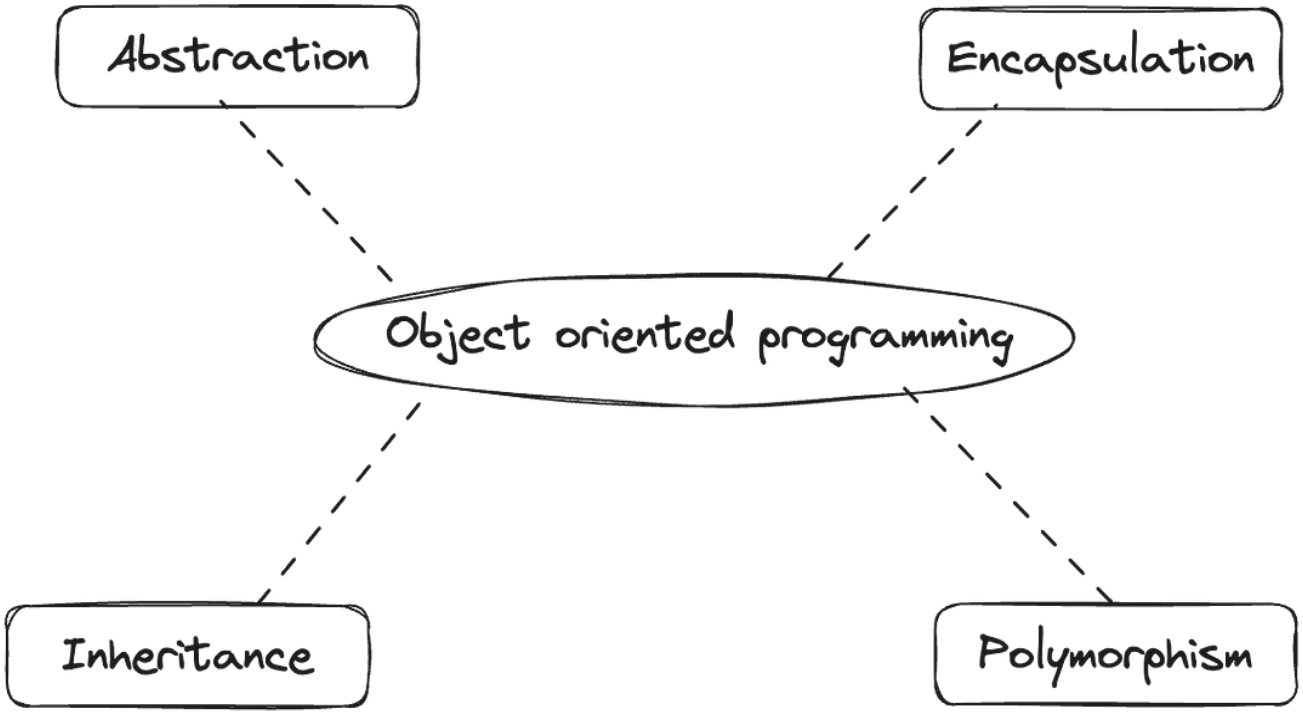
```
// Reverse an array
//Use the Array.Reverse() method to reverse the order of elements.
int[] numbers = { 10, 20, 30, 40, 50 };

Array.Reverse(numbers);

foreach (int number in numbers)
{
    Console.Write(number + " "); // Output: 50 40 30 20 10
}
```

```
// Searching an array
//Use the Array.IndexOf() method to find the index
//of a specific element.
//If the element is not found, it returns -1.
int[] numbers = { 10, 20, 30, 40, 50 };

int index = Array.IndexOf(numbers, 30);
Console.WriteLine("Index of 30: " + index); // Output: 2

index = Array.IndexOf(numbers, 60);
Console.WriteLine("Index of 60: " + index); // Output: -1 (not found)

```

Access modifier:

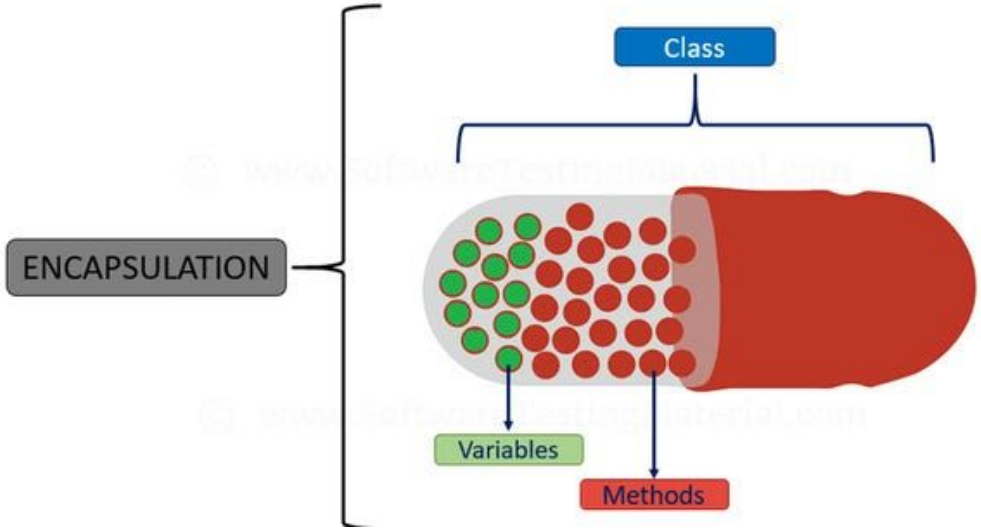
- Public → Accessible from anywhere
- Private → Accessible only within the class
- Protected → Accessible within the class and its derived class
- Internal → Accessible within the same assembly
- Protected Internal → Accessible within the same assembly and derived class

Main Pillars of Object-Oriented Programming :



Encapsulation

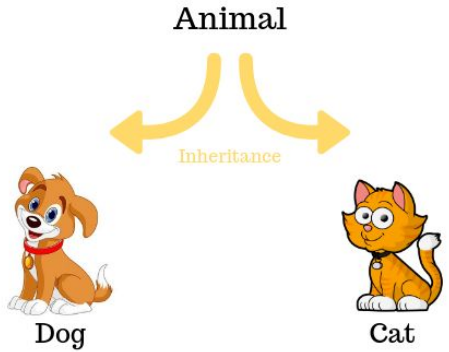
- Wrapping up a data member and a method together.
- Hiding and protecting data
- Controlled Access



Inheritance

➢ **Should satisfy the IS-A test.**

- **Superclass (Parent Class/ Base Class)**
- **Subclass (Child Class/ Derived Class)**



Types of Inheritance in C#

- Single Inheritance → A child class inherits from one parent.
- Multilevel Inheritance → A child class inherits from another child.
- Hierarchical Inheritance → Multiple classes inherit from one parent.
- Interface Inheritance → A class implements multiple interfaces.

Polymorphism

- Greek word which means many forms
- Compile-Time Polymorphism (Static)
- Run-Time Polymorphism (Dynamic)

Abstraction

Hiding complex implementation details and showing only the essential information to the user

Abstraction is typically achieved through two mechanisms:

- **Abstract Classes**
- **Interfaces**