

2015



**C#/ .NET**  
**programming basics**

The logo features a stylized profile of a human head in grey, with a yellow brain inside. The brain is depicted with white lines representing neural connections.

# **BASIC PRINCIPLES OF C#, CLR**

BRAIN  
ACADEMY

# Training program

- **Block 1.** C# programming fundamentals
- **Block 2.** Windows application development
- **Block 3.** Service-oriented and web application development
- **Block 4.** Application architecture and design patterns
- **Block 5.** Certification

# Block 1 contents

1. Basic principles of C#, CLR
2. Object oriented fundamentals
3. Exception handling
4. Advanced programming (Delegates, events, lambdas. Generics. Collections)
5. Assembly management and application debug
6. Multithreading and asynchronous processing
7. Data access
8. Unsafe code and pointers. .Net Framework security

# Array, structure, enum

- Basic principles of C#, CLR
  - C# & CLR basics
  - Data types
  - Operators
  - Array, Structure, Enum
  - System.Console


# Lecture contents

- Array, Structure, Enum
  - Array. Array size
  - Arrays (single dimensional, multi dimensional, jagged)
  - Array operations (comparison, searching, sorting)
  - Enumerations
  - Structures

# Array. Array size

- Array, Structure, Enum
  - Array. Array size
  - Arrays (single dimensional, multi dimensional, jagged)
  - Array operations (comparison, searching, sorting)
  - Enumerations
  - Structures

# Array

 **Array** – in programming, a list of data values, all of the same type, any element of which can be referenced by an expression consisting of the array name followed by an indexing expression. Arrays are part of the fundamentals of data structures, which, in turn, are a major fundamental of computer programming.



# Array size

- Array consist of several elements
- Size of the array is not part of its type
- It is possible to declare an array and assign any array of elements to it, regardless of the array's length
- It is necessary to define size of array
- If a specified array index is out of bounds, an **IndexOutOfRangeException** is thrown



# Array example

- `//array definition`
- `int [] int_array = new int [5];`
- `// array initialization`
- `int_array[0] = 1;`
- `int_array[1] = 2;`
- `int_array[2] = 3;`

# Arrays (single dimensional, multi dimensional, jagged)

- Array, Structure, Enum
  - Array. Array size
  - Arrays (single dimensional, multi dimensional, jagged)
  - Array operations (comparison, searching, sorting)
  - Enumerations
  - Structures

# Single dimensional array

- C# supports single dimensional arrays, multi dimensional arrays (rectangular arrays), and array-of-arrays (jagged arrays).
- Single dimensional array – is the simplest form of array

# Multi dimensional array

- Multi dimensional array has several dimensions
- Use comas to define number of dimensions:
  - 2 dimensions: `int[,] some_array;`
  - 3 dimensions: `int[,,] some_array;;`
  - And so on...

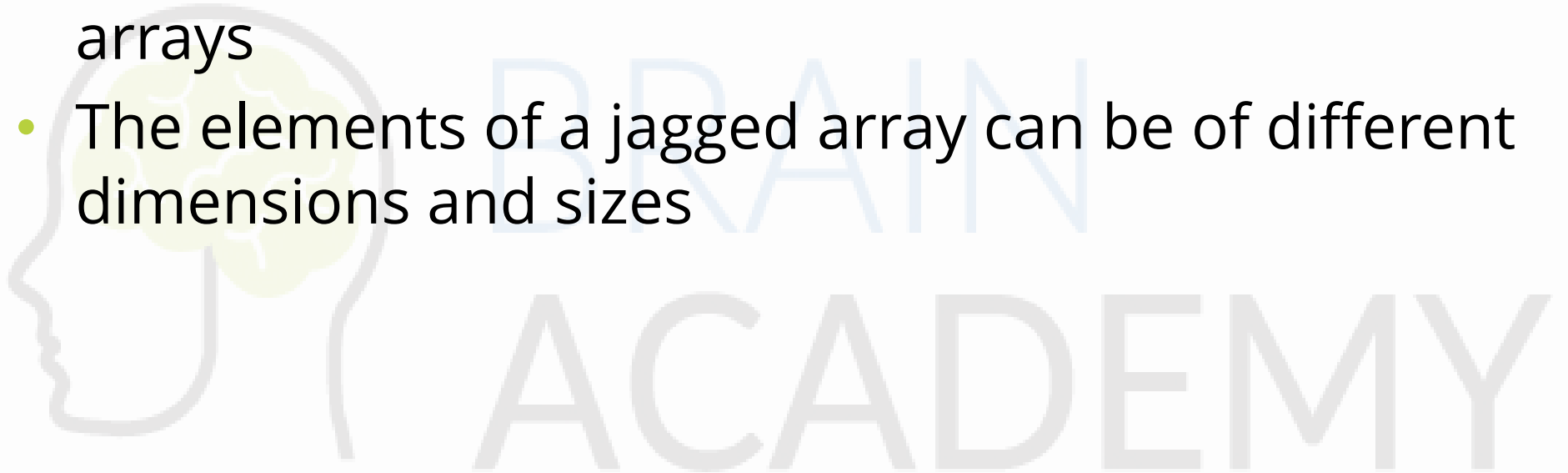


## Multi dimensional array example

1. `//array definition`
2. `int[,] int_two_dim_jag_array = new int[2, 2];`
3. `// array initialization`
4. `int_two_dim_jag_array[0, 0] = 1;`
5. `int_two_dim_jag_array[0, 1] = 2;`
6. `int_two_dim_jag_array[1, 0] = 3;`
7. `int_two_dim_jag_array[1, 1] = 4;`

# Jagged array

- A jagged array is an array whose elements are arrays
- The elements of a jagged array can be of different dimensions and sizes





# Jagged array example

1. `//array definition`
2. `int[][] int_jagged_array = new int[3][];`
3. `// array initialization`
4. `int_jagged_array[0] = new int[2] {2,3};`
5. `int_jagged_array[1] = new int[3] {5,3,7};`
6. `int_jagged_array[2] = new int[2] {8,1};`



# Some array properties and methods

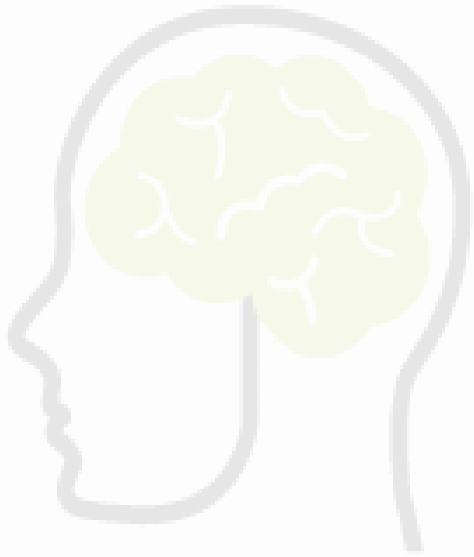
- **Length** - returns a 32-bit integer that represents the total number of items in all the dimensions of an array
- **Rank** - Returns the number of dimensions of an array
- **GetLength()** - returns the number of items in an array

# Array operations (comparison, searching, sorting)

- Array, Structure, Enum
  - Array. Array size
  - Arrays (single dimensional, multi dimensional, jagged)
  - Array operations (comparison, searching, sorting)
  - Enumerations
  - Structures

# Array operations (searching) (1/2)

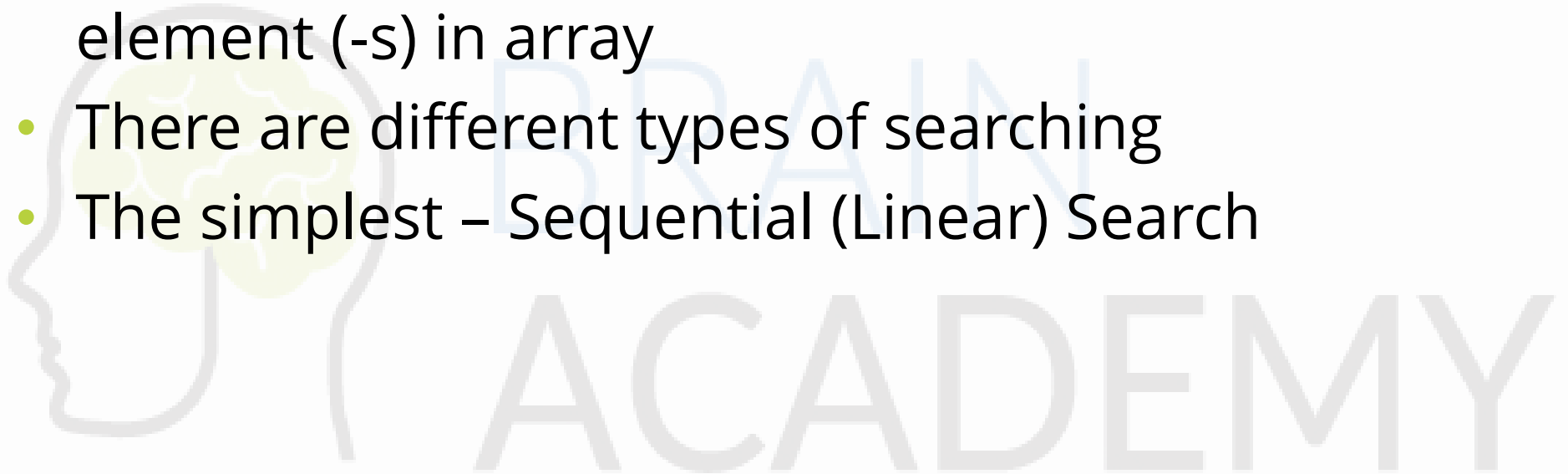
 **Search** - to try to find someone or something



BRAIN  
ACADEMY

## Array operations (searching) (2/2)

- Searching in arrays is process of finding some element (-s) in array
- There are different types of searching
- The simplest – Sequential (Linear) Search




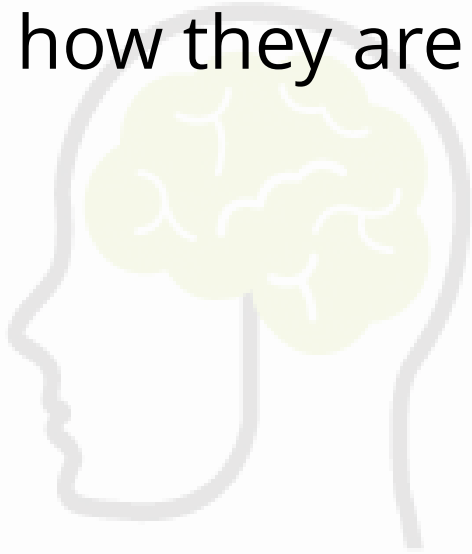


## Array operations (searching) example

```
1. // suggest value will not be find
2. int position = -1;
3. // searching of position of last element
   equals 2
4. for (int i = 0; i < int_array.Length; i++)
5.     if (int_array[i] == 2)
6.         position = i;
```

# Array operations (comparison) (1/2)

 **Comparison** - the act of looking at things to see how they are similar or different



BRAIN  
ACADEMY

## Array operations (comparison) (2/2)

- Array comparison is used to compare 2 or more arrays: are the equals or aren't
- There are 2 types of comparison:
  - Reference comparison
  - Value comparison
- In typical value comparison size of arrays should be the same




# Array operations (comparison) example

```
1. // suggest arrays are equals
2. bool flag = true;
3. // arr1==arr2 (without size checking)
4. for (int i = 0; i < arr1.GetLength(0);
    i++)
5.     for (int j = 0; j < arr2.GetLength(1);
        j++)
6.         if (arr1[i, j] != arr2[i, j])
7.             flag = false;
```



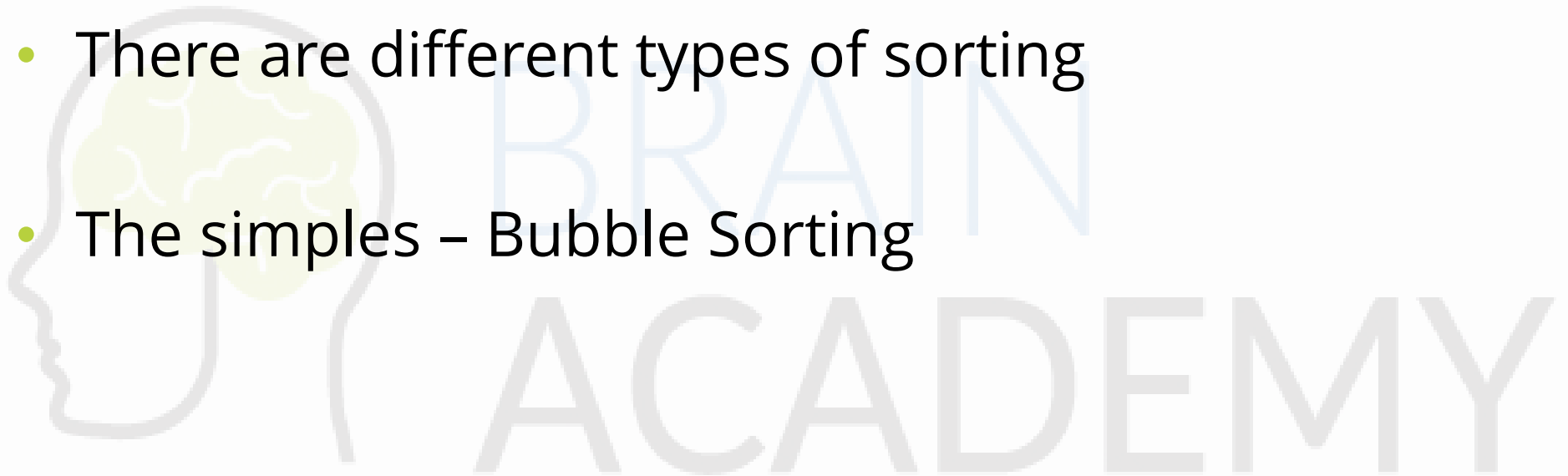
# Array operations (sorting) (1/2)

 **Sorting** - a method of arranging data based on the order of specified information.

 **Sorting order** - a way to arrange data based on value or data type.

# Array operations (sorting) (2/2)

- There are different types of sorting
- The simplest – Bubble Sorting





## Array operations (sorting) example

```
1. int temp = 0;
2. for (int i = 0; i < int_array.Length; i++)
3.     for (int j = 0; j < int_array.Length -
4.         1; j++)
5.         if (int_array[j] > int_array[j + 1])
6.             {
7.                 temp = int_array[j + 1];
8.                 int_array[j + 1] = int_array[j];
9.                 int_array[j] = temp;
10.            }
```

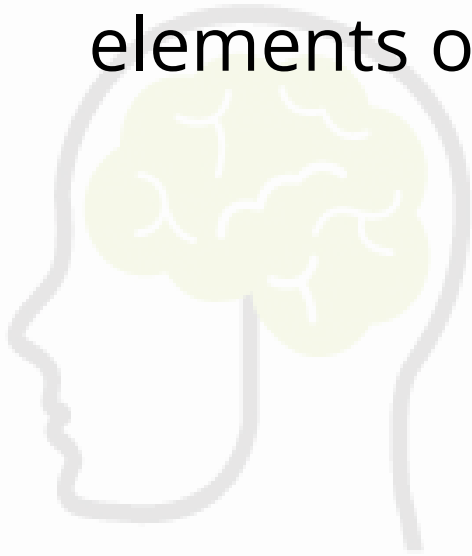
## Copy of array (1/2)



**Copy** - to duplicate information and reproduce it in another part of a document, in a different file or memory location, or in a different medium. A copy operation can affect data ranging from a single character to large segments of text, a graphics image, or from one to many data files.

## Copy of array (2/2)

- Copy of array is process of copying all the elements of an array to another array



BRAIN  
ACADEMY



## Copy of array example

```
1. // arr1==arr2 (without size checking)
2. for (int i = 0; i < arr1.GetLength(0);
   i++)
3.     for (int j = 0; j < arr2.GetLength(1);
   j++)
4.         arr2[i][j] = arr1[i][j];
```

# Array summary

- There are single dimensional, multi dimensional, jagged arrays
- Arrays are used to group data of the same type
- Main operations with arrays are:
  - Searching
  - Sorting
  - Comparison
  - Copying

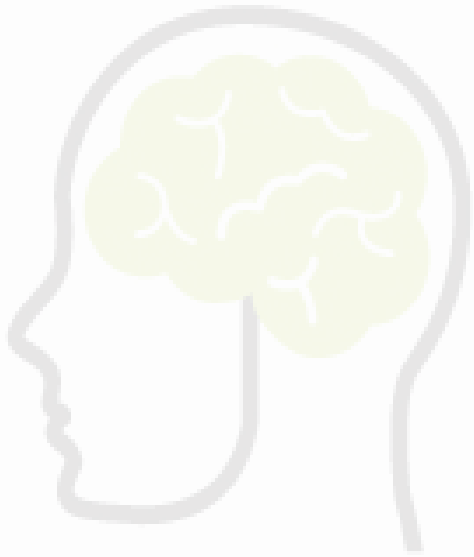
# Enumerations

- Array, Structure, Enum
  - Array. Array size
  - Arrays (single dimensional, multi dimensional, jagged)
  - Array operations (comparison, searching, sorting)
  - Enumerations
  - Structures



# Enumerations (1/5)

 **Enumeration** - a list of named constants.



BRAIN  
ACADEMY

## Enumerations (2/5)

- The **enum** keyword is used to declare an enumeration
- By default, the first enumerator has the value 0, and the value of each successive enumerator is increased by 1.
- Example
- `enum Marks {Unsatisfactory, Satisfactory, Good, Excellent};`

## Enumerations (3/5)

- Enumerators can use initializers to override the default values
- Example
- `enum Marks {Unsatisfactory = 2, Satisfactory, Good, Excellent};`

## Enumerations (4/5)

- Every enumeration type has an underlying type, which can be any integral type except **char**. The default underlying type of enumeration elements is **int**.
- To declare an enum of another integral type, such as **byte**, use a colon after the identifier followed by the type
- `enum Marks : byte {Unsatisfactory = 2, Satisfactory, Good, Excellent};`


## Enumerations (5/5)

- The underlying type specifies how much storage is allocated for each enumerator. However, an explicit cast is necessary to convert from **enum** type to an integral type.
- `int mark = (int)Marks.Excellent;`
- Every enumeration is a type, declared by user. It is reference type.

# Structures

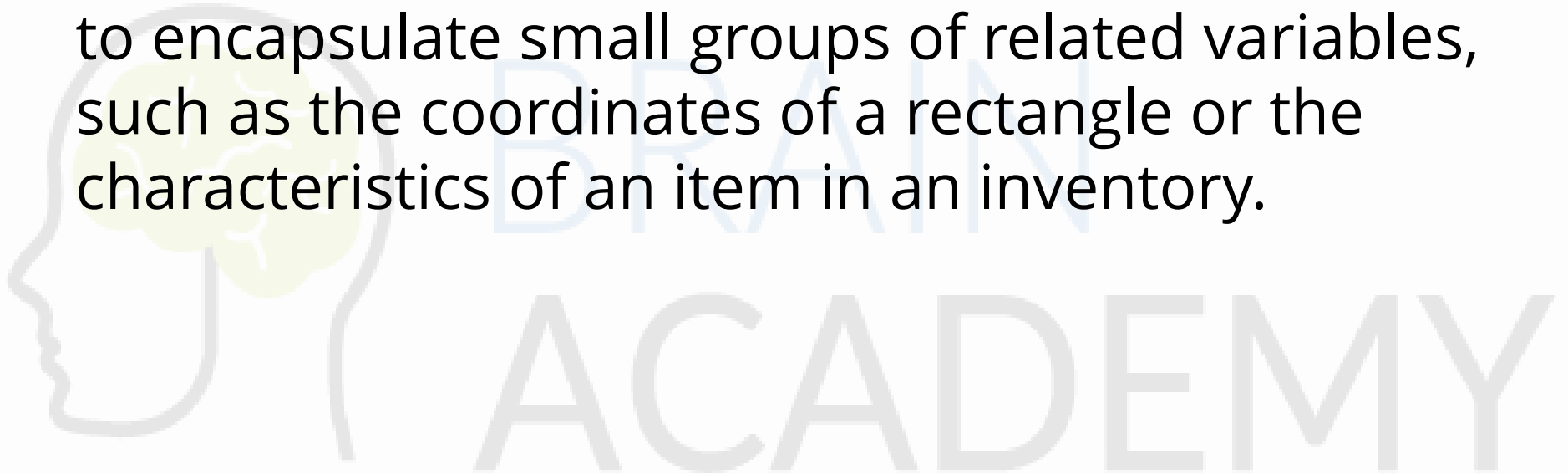
- Array, Structure, Enum
  - Array. Array size
  - Arrays (single dimensional, multi dimensional, jagged)
  - Array operations (comparison, searching, sorting)
  - Enumerations
  - Structures

## Structures (1/2)

 **Struct** - a compound data type that is typically used to contain a few variables that have some logical relationship. Structs can also contain methods and events. Structs do not support inheritance but they do support interfaces. A struct is a value type, while a class is a reference type.

## Structures (2/2)

- A **struct** type is a value type that is typically used to encapsulate small groups of related variables, such as the coordinates of a rectangle or the characteristics of an item in an inventory.





# Structure declaration example

```
1. //struct Animal declaration
2. struct Animal
3. {
4.     public string species;
5.     public string name;
6.     public int age;
7.     public Sex sex;
8. }
9. enum Sex { Male, Female };
```



# Structure objects example

```
1. //Declare and initialize struct Animal objects
2.     Animal lion;
3.     lion.species = "lion";
4.     lion.name = "leo";
5.     lion.age = 7;
6.     lion.sex = Sex.Male;

7.     Animal lion2 = new Animal();
8.     lion2.species = lion.species;
9.     lion2.name = "King";
10.    lion2.age = 5;
11.    lion2.sex = Sex.Male;
```

# Enums and structs summary

- **enum** and **struct** make possible to declare user types; and then declare variables of these types
- They are reference types
- Structs can also contain constructors, constants, fields, methods, properties, indexers, operators, events, and nested types
- It will be explained in next module