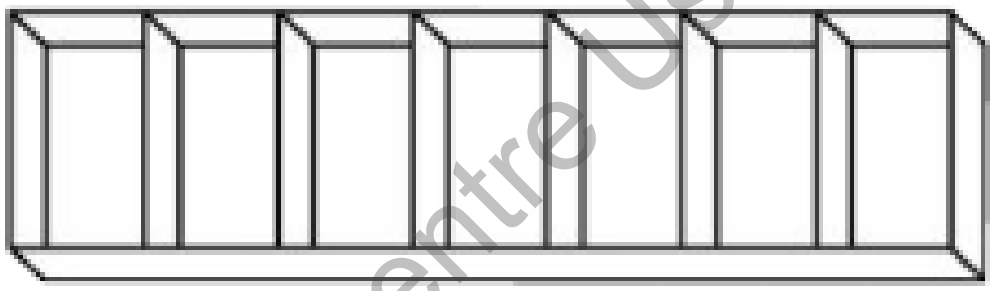
Session: 5

Arrays

Objectives

- Define and describe arrays
- List and explain the types of arrays
- Explain the Array class

- An array is a collection of elements of a single data type stored in adjacent memory locations.



Example

- In a program, an array can be defined to contain 30 elements to store the scores of 30 students.

Example

- Consider a program that stores the names of 100 students.
- To store the names, the programmer would create 100 variables of type string.
- Creating and managing these 100 variables is a tedious task as it results in inefficient memory utilization.
- In such situations, the programmer can create an array for storing the 100 names.

Array of 100 Names

Steve	David	John	Klen	Stefen	.....

Proper Utilization of Memory

100 Variables Storing Names

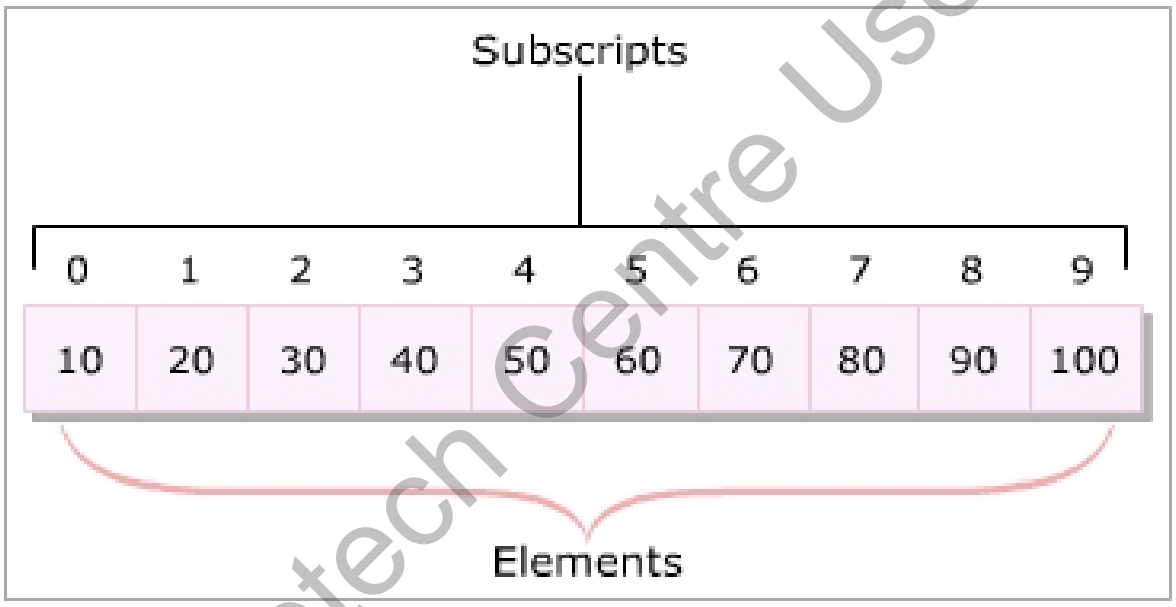
Program to store 100 names of students	
var empOne	Steve
var studentTwo	David
var studentThree	John
var studentFour	Klen
var studentFive	Stefen
…	…
… Till 100 variables	

Inefficient Memory Utilization

- An array:
  - Is a collection of related values placed in contiguous memory locations and these values are referenced using a common array name.
  - Simplifies the task of maintaining these values.

- An array always stores values of a single data type.
- Each value is referred to as an element.
- These elements are accessed using subscripts or index numbers that determine the position of the element in the array list.
- C# supports zero-based index values in an array.
- This means that the first array element has an index number zero while the last element has an index number n-1, where n stands for the total number of elements in the array.
- This arrangement of storing values helps in efficient storage of data, easy sorting of data, and easy tracking of the data length.

- Following figure is an example of the subscripts and elements in an array:



Arrays

Declaring Arrays

- Arrays are reference type variables whose creation involves two steps:
  - **Declaration:**
    - An array declaration specifies the type of data that it can hold and an identifier.
    - This identifier is basically an array name and is used with a subscript to retrieve or set the data value at that location.
  - **Memory allocation:**
    - Declaring an array does not allocate memory to the array.
- Following is the syntax for declaring an array:

Syntax

```
type[] arrayName;
```

- In the syntax:
  - `type`: Specifies the data type of the array elements (for example, `int` and `char`).
  - `arrayName`: Specifies the name of the array.

- An array can be:
  - Created using the `new` keyword and then initialized.
  - Initialized at the time of declaration itself, in which case the `new` keyword is not used.
- Creating and initializing an array with the `new` keyword involves specifying the size of an array.
- The number of elements stored in an array depends upon the specified size.
- The `new` keyword allocates memory to the array and values can then be assigned to the array.

- If the elements are not explicitly assigned, default values are stored in the array.
- The following table lists the default values for some of the widely used data types:

Data Types	Default Values
`int`	0
`float`	0.0
`double`	0.0
`char`	'\0'
`string`	`Null`

- The following syntax is used to create an array:

Syntax

```
arrayName = new type[size-value];
```

- The following syntax is used to declare and create an array in the same statement using the `new` keyword:

Syntax

```
type[] arrayName = new type[size-value];
```

- In the syntax:
  - `size-value`: Specifies the number of elements in the array. You can specify a variable of type `int` that stores the size of the array instead of directly specifying a value.

- Once an array has been created using the syntax, its elements can be assigned values using either a subscript or using an iteration construct such as a `for` loop.

- The following syntax is used to create and initialize an array without using the `new` keyword:

Syntax

```
type[ ] arrayIdentifier = {val1, val2, val3, ..., valN};
```

- In the syntax:
  - `val1`: It is the value of the first element.
  - `valN`: It is the value of the nth element.

- The following code creates an integer array which can have a maximum of five elements in it:

Snippet

```
public int[] number = new int[5];
```

- The following code initializes an array of type `string` that assigns names at appropriate index locations:

Snippet

```
public string[] studNames = new string{“Allan”, “Wilson”,
“James”, “Arnold”};
```

- In the code:
  - The string '`Allan`' is stored at subscript 0, '`Wilson`' at subscript 1, '`James`' at subscript 2, and '`Arnold`' at subscript 3.

- The following code stores the string 'Jack' as the name of the fifth enrolled student:

Snippet

```
studNames[4] = "Jack";
```

- The following code demonstrates another approach for creating and initializing an array. An array called **count** is created and is assigned int values:

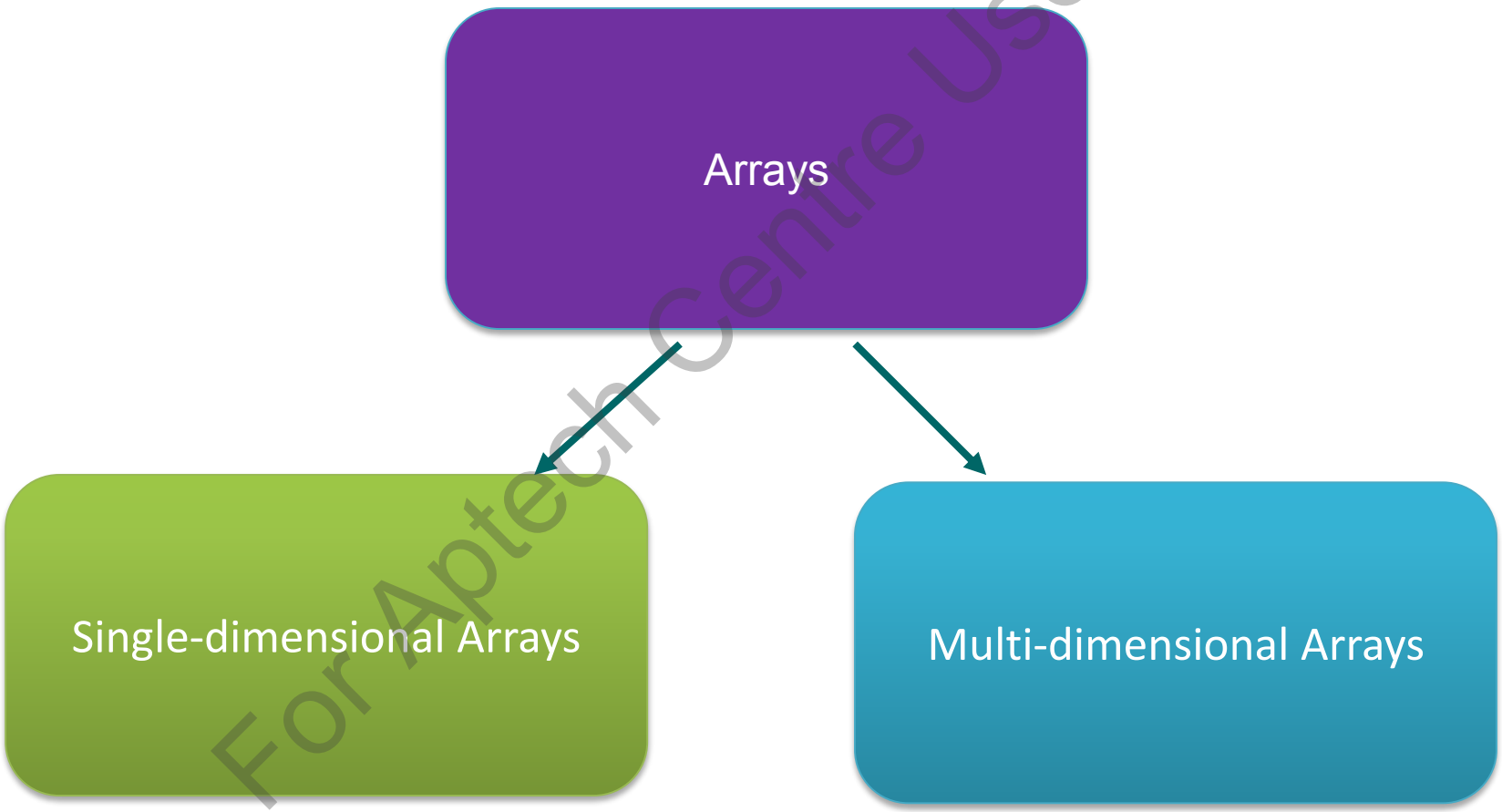
```
using System;
class Numbers
{
   static void Main(string[] args)
   {
      int[] count = new int[10];//array is created
      int counter = 0;
      for(int i = 0; i< 10; i++)
      {
       count[i] = counter++; //values are assigned to the elements
      Console.WriteLine("The count value is: " + count[i]);
      //element values are printed
      }
     }
}
```

- In the code:
  - The class **Numbers** declares an array variable **count** of size 10.
  - An `int` variable **counter** is declared and is assigned the value 0.
  - Using the `for` loop, every element of the array **count** is assigned the incremented value of the variable **counter**.

Output

```
The count value is: 0
The count value is: 1
The count value is: 2
The count value is: 3
The count value is: 4
The count value is: 5
The count value is: 6
The count value is: 7
The count value is: 8
The count value is: 9
```

- Based on how arrays store elements, arrays can be categorized into following two types:



- Single-dimensional arrays:
  - Elements of a single-dimensional array stored in a single row in allocated memory.
  - Declaration/initialization same as standard declaration/initialization of arrays.
  - Elements indexed from 0 to (n-1), where n is the total number of elements in the array.

Example

Syntax

- The following syntax is used for declaring and initializing a single-dimensional array:

```
type[] arrayName; //declaration
arrayName = new type[length]; // creation
```

- In the syntax:
  - `type`: Is a variable type and is followed by square brackets ([]).
  - `arrayName`: Is the name of the variable.
  - `length`: Specifies the number of elements to be declared in the array.
  - `new`: Instantiates the array.

- The following code initializes a single-dimensional array to store the name of students:

```
using System;

classSingleDimensionArray
{
     static void Main(string[] args)
     {
          string[] students = new string[3] {"James", "Alex", "Fernando"};
          for (int i=0; i<students.Length; i++)
          {
               Console.WriteLine(students[i]);
          }
     }
}
```

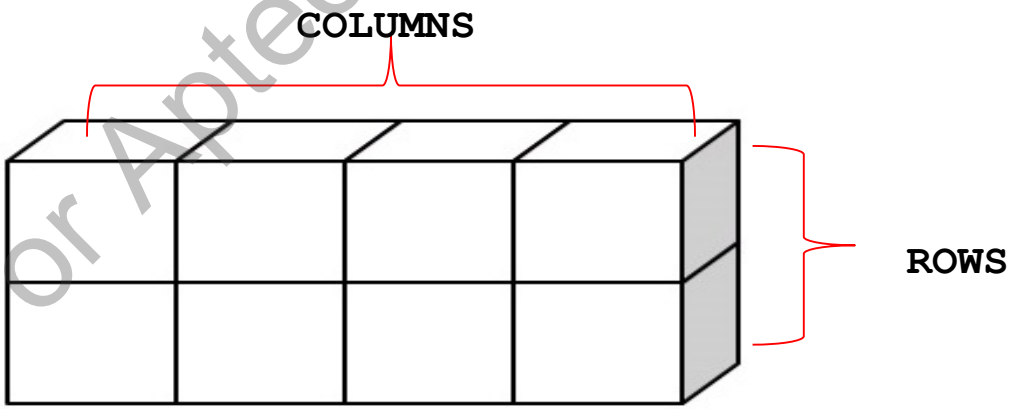
- In the code:
  - The class **`SingleDimensionArray`** stores the names of the students in the students array.
  - An integer variable `i` is declared in the `for` loop that indicates the total number of students to be displayed.
  - Using the `for` loop, the names of the students are displayed as the output.

Output

```
James
Alex
Fernando
```

Example

- Consider a scenario where you need to store the roll numbers of 50 students and their marks in three exams.
- Using a single-dimensional array, you require two separate arrays for storing roll numbers and marks respectively.
- However, using a multi-dimensional array, you just need one array to store both roll numbers as well as marks.



- A multi-dimensional array allows you to store combination of values of a single type in two or more dimensions.
- The dimensions of the array are represented as rows and columns similar to the rows and columns of a Microsoft Excel sheet.
- Following are the two types of multi-dimensional arrays:

Rectangular Array	• Is a multi-dimensional array where all the specified dimensions have constant values. • Will always have the same number of columns for each row.
Jagged Array	• Is a multidimensional array where one of the specified dimensions can have varying sizes. • Can have unequal number of columns for each row.

- The following is the syntax for creating a rectangular array:

Syntax

```
type[,] <arrayName>; //declaration
arrayName = new type[value1 , value2]; //initialization
```

- In the syntax:
  - `type`: Is the data type and is followed by `[]`.
  - `arrayName`: Is the name of the array.
  - `value1`: Specifies the number of rows.
  - `value2`: Specifies the number of columns.

- The following code demonstrates the use of rectangular arrays:

Snippet

```
using System;
classRectangularArray
{
static void Main (string [] args)
{
      int[,] dimension = new int [4, 5];
      intnumOne = 0;
      for (int i=0; i<4; i++)
      {
            for (int j=0; j<5; j++)
            {
            dimension [i, j] = numOne;
            numOne++;
            }
      }
      for (int i=0; i<4; i++)
      {
      for (int j=0; j<5; j++)
      {
      Console.Write(dimension [i, j] + " ");
      }
      Console.WriteLine();
      }
      }
}
```

- In the code:
  - A rectangular array called `dimension` is created that will have four rows and five columns.
  - The `int` variable **`numOne`** is initialized to zero.
  - The code uses nested `for` loops to store each incremented value of **`numOne`** in the `dimension` array.
  - These values are then displayed in the matrix format using again the nested `for` loops.

Output

```
0  1  2  3  4
5  6  7  8  9
10  11  12  13  14
15  16  17  18  19
```

- Arrays can be either:

Fixed-length arrays

The number of elements is defined at the time of declaration.

For example, an array declared for storing days of the week will have exactly seven elements.

The number of elements is known and hence, can be defined at the time of declaration. Therefore, a fixed-length array can be used.

Dynamic arrays

The size of the array is not fixed at the time of the array declaration and can dynamically increase at runtime or whenever required.

For example, an array declared to store the e-mail addresses of all users who access a particular Web site cannot have a predefined length.

In such a case, the length of the array cannot be specified at the time of declaration and a dynamic array has to be used.

Can add more elements to the array as and when required.

Created using built-in classes of the .NET Framework.

- The following code demonstrates the use of fixed arrays:

```
using System;
classDaysofWeek
{
     static void Main(string[] args)
     {
          string[] days = new string[7];
          days[0] = "Sunday";
          days[1] = "Monday";
          days[2] = "Tuesday";
          days[3] = "Wednesday";
          days[4] = "Thursday";
          days[5] = "Friday";
          days[6] = "Saturday";
          for(int i = 0; i<days.Length; i++)
          {
          Console.WriteLine(days[i]);
          }
     }
}
```

- In this code:
  - A fixed-length array variable, **`days`**, of data type `string`, is declared to store the seven days of the week.
  - The days from Sunday to Saturday are stored in the index positions 0 to 6 of the array and are displayed on the console using the `Console.WriteLine()` method.

Output

- The following output displays the use of fixed arrays:

Array References 1-2

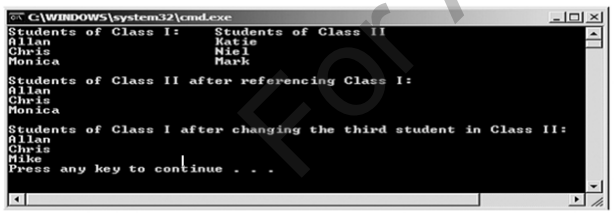
- An array variable can be referenced by another array variable (referring variable).
- While referring, the referring array variable refers to the values of the referenced array variable.
- The following code demonstrates the use of array references:

```
using
using System;
classStudentReferences
{
     public static void Main()
     {
     string[] classOne = { "Allan", "Chris", "Monica" };
     string[] classTwo = { "Katie", "Niel", "Mark"};
     Console.WriteLine("Students of Class I:\tStudents of Class II");
     for (int i = 0; i< 3; i++)
     {
       Console.WriteLine(classOne[i] + "\t\t\t" + classTwo[i]);
     }

     classTwo = classOne;
     Console.WriteLine("\nStudents of Class II after referencing
     Class I:");
     for (int i = 0; i< 3; i++)
     {
```

```
        Console.WriteLine(classTwo[i] + " ");
      }
      Console.WriteLine();
      classTwo[2] = "Mike";
      Console.WriteLine("Students of Class I after changing the third
      student in Class II:");
      for (int i = 0; i< 3; i++)
      {
         Console.WriteLine(classOne[i] + " ");
      }
}
}
```

- In the code:
  - **classOneis** assigned to **classTwo;** therefore, both the arrays reference the same set of values.
  - Consequently, when the third array element of **classTwo** is changed from 'Monica' to 'Mike', an identical change is seen in the third element of **classOne**.
- The following figure displays the use of array references:



- A rectangular array is a two-dimensional array where each row has an equal number of columns.
- The following syntax displays the marks stored in a rectangular array:

Syntax

```
type [,]<variableName>;
variableName = new type[value1 , value2];
```

- In the syntax:
  - `type`: Specifies the data type of the array elements.
  - `[,]`: Specifies that the array is a two-dimensional array.
  - `variableName`: Specifies the name of the two-dimensional array.
  - `new`: Is the operator used to instantiate the array.
  - `value1`: Specifies the number of rows in the two-dimensional array.
  - `value2`: Specifies the number of columns in the two-dimensional array.

- The following code allows the user to specify the number of students, their names, the number of exams, and the marks scored by each student in each exam.
- All these marks are stored in a rectangular array.

Snippet

```
using System;
class StudentsScore
{
   void StudentDetails()
   {
      Console.Write("Enter the number of Students: ");
      int noOfStds = Convert.ToInt32(Console.ReadLine());
      Console.Write("Enter the number of Exams: ");
      int exams = Convert.ToInt32(Console.ReadLine());
      string[] stdName = new string[noOfStds];
      string[,] details = new string[noOfStds, exams];

      for (int i = 0; i<noOfStds; i++)
      {
      Console.WriteLine();
      Console.Write("Enter the Student Name: ");
      stdName[i] = Convert.ToString(Console.ReadLine());
      for (int y = 0; y < exams; y++)
      {
      Console.Write("Enter Score in Exam " + (y + 1) + ": ");
      details[i, y] = Convert.ToString(Console.ReadLine());
```

Snippet

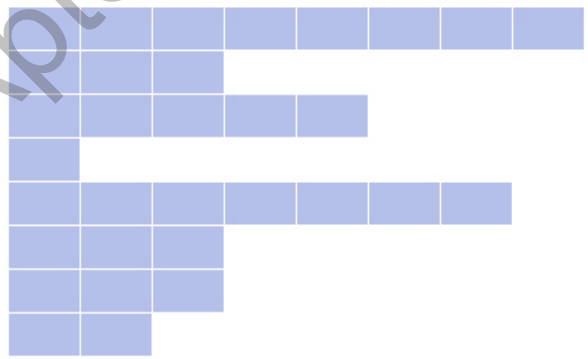
```
        }
        }
        Console.WriteLine();
        Console.WriteLine("Student Exam Details");
        Console.WriteLine("--------------------");
        Console.WriteLine();
        Console.WriteLine("Student\t\tMarks");
        Console.WriteLine("-----\t\t------" );
        for (int i = 0; i<stdName.Length; i++)
        {
        Console.WriteLine(stdName[i]);
        for (int j = 0; j < exams; j++)
        {
        Console.WriteLine("\t\t" + details[i, j]);
        }
        Console.WriteLine();
    }
    }
    static void Main()
    {
    StudentsScore objStudentsScore = new StudentsScore();
    objStudentsScore.StudentDetails();
    }
}
```

- In the code:
  - The **StudentsScore** class allows the user to enter the number of students in the class, the names of the students, the number of exams conducted, and the marks scored by each student in each exam.
  - The class declares a method **StudentDetails**, which accepts the student and the exam details.
  - The variable **noOfStds** stores the number of students whose details are to be stored.
  - The variable **exams** stores the number of exams the students have appeared in. The array **stdName** stores the names of the students.
  - The dimensions of the rectangular array **details** are defined by the variables **noOfStds** and **exams**.
  - This array stores the marks scored by students in the various exams. A nested `for` loop is used for displaying the student details.
  - In the `Main` method, an object is created of the class **StudentsScore** and the method **StudentDetails** is called through this object.

- A jagged array:
  - Is a multi-dimensional array and is referred to as an array of arrays.
  - Consists of multiple arrays where the number of elements within each array can be different. Thus, rows of jagged arrays can have different number of columns.
  - Optimizes the memory utilization and performance because navigating and accessing elements in a jagged array is quicker as compared to other multi-dimensional arrays.

Example

- Consider a class of 500 students where each student has opted for a different number of subjects.
- Here, you can create a jagged array because the number of subjects for each student varies.
- The following figure displays the representation of jagged arrays:



- The following code demonstrates the use of jagged arrays to store the names of companies:

```
using System;

classJaggedArray
{
    static void Main (string[] args)
    {
       string[][] companies = new string[3][];
       companies[0] = new string[] {"Intel", "AMD"};
       companies[1] = new string[] {"IBM", "Microsoft", "Sun"};
       companies[2] = new string[] {"HP", "Canon", "Lexmark",
"Epson"};
       for (int i=0; i<companies.GetLength (0); i++)
       {
       Console.Write("List of companies in group " + (i+1) +
":\t");
       for (int j=0; j<companies[i].GetLength (0); j++)
       {
       Console.Write(companies [i][j] + " ");
       }
       Console.WriteLine();
       }
    }

}
```

- In the code:
  - A jagged array called **`companies`** is created that has three rows.
  - The values '`Intel`' and '`AMD`' are stored in two separate columns of the first row.
  - Similarly, the values '`IBM`', '`Microsoft`', and '`Sun`' are stored in three separate columns of the second row.
  - Finally, the values '`HP`', '`Canon`', '`Lexmark`', and '`Epson`' are stored in four separate columns of the third row.

Output

```
List of companies in group 1: Intel AMD
List of companies in group 2: IBM Microsoft Sun
List of companies in group 3: HP Canon Lexmark Epson
```

Using the foreach Loop for Arrays 1-3

- The foreach loop:
  - In C# is an extension of the for loop.
  - Is used to perform specific actions on large data collections and can even be used on arrays.
  - Reads every element in the specified array.
  - Allows you to execute a block of code for each element in the array.
  - Is particularly useful for reference types, such as strings.

- The following is the syntax for the foreach loop:

Syntax

```
foreach(type<identifier> in <list>)
{
// statements
}
```

- In the code:
  - type: Is the variable type.
  - identifier: Is the variable name.
  - list: Is the array variable name.

- The following code displays the name and the leave grant status of each student using the `foreach` loop:

Snippet

```
using System;

class Students
{
   static void Main(string[] args)
   {
    string[] studentNames = new string[3] { “Ashley”, “Joe”,
    “Mikel”};
    foreach (string studName in studentNames)
    {
      Console.WriteLine(“Congratulations!! “ + studName + “ you
       have been granted an extra leave”);
    }
   }
}
```

Using the foreach Loop for Arrays 3-3

- In the code:
  - The **Students** class initializes an array variable called **studentNames**.
  - The array variable **studentNames** stores the names of the students.
  - In the `foreach` loop, a `string` variable **studName** refers to every element stored in the array variable **studentNames**.
  - For each element stored in the **studentNames** array, the `foreach` loop displays the name of the student and grants a day's leave extra for each student.

Output

```
Congratulations!! Ashley you have been granted an extra leave
Congratulations!! Joe you have been granted an extra leave
Congratulations!! Mikel you have been granted an extra leave
```

- The `Array` class:
  - Is a built-in class in the `System` namespace and is the base class for all arrays in C#.
  - Provides methods for various tasks such as creating, searching, copying, and sorting arrays.

Example

- Consider a code that stores the marks of a particular subject for 100 students.
- The programmer wants to sort the marks, and to do this, he/she has to manually write the code to perform sorting.
- This can be tedious and result in increased lines of code.
- However, if the array is declared as an object of the `Array` class, the built-in methods of the `Array` class can be used to sort the array.

- The `Array` class consists of system-defined properties and methods that are used to create and manipulate arrays in C#.
- The properties are also referred to as system array class properties.
  - **Properties:**
    - The properties of the `Array` class allow you to modify the elements declared in the array.
    - The following table displays the properties of the `Array` class:

Properties	Descriptions
`IsFixedSize`	Returns a boolean value, which indicates whether the array has a fixed size or not. The default value is true.
`IsReadOnly`	Returns a boolean value, which indicates whether an array is read-only or not. The default value is false.
`IsSynchronized`	Returns a boolean value, which indicates whether an array can function well while being executed by multiple threads together. The default value is false.
`Length`	Returns a 32-bit integer value that denotes the total number of elements in an array.
`LongLength`	Returns a 64-bit integer value that denotes the total number of elements in an array.
`Rank`	Returns an integer value that denotes the rank, which is the number of dimensions in an array.
`SyncRoot`	Returns an object which is used to synchronize access to the array.

- **Methods:**
  - The `Array` class allows you to clear, copy, search, and sort the elements declared in the array.
  - The following table displays the most commonly used methods in the `Array` class:

Methods	Descriptions
`Clear`	Deletes all elements within the array and sets the size of the array to 0.
`CopyTo`	Copies all elements of the current single-dimensional array to another single-dimensional array starting from the specified index position.
`GetLength`	Returns number of elements in an array.
`GetLowerBound`	Returns the lower bound of an array.
`GetUpperBound`	Returns the upper bound of an array.
`Initialize`	Initializes each element of the array by calling the default constructor of the Array class.
`Sort`	Sorts the elements in the single-dimensional array.
`SetValue`	Sets the specified value at the specified index position in the array.
`GetValue`	Gets the specified value from the specified index position in the array.

Using the Array Class 1-5

- The `Array` class allows you to create arrays using the `CreateInstance()` method.
- This method can be used with different parameters to create single-dimensional and multi-dimensional arrays.
- For creating an array using this class, you need to invoke the `CreateInstance()` method that is accessed by specifying the class name because the method is declared as static.
- The following is the syntax for signature of the `CreateInstance()` method used for creating a single-dimensional array:

Syntax

```
public static Array CreateInstance(Type elementType, int length)
```

- In the syntax:
  - `Array`: Returns a reference to the created array.
  - `Type`: Uses the `typeof` operator for explicit casting.
  - `elementType`: Is the resultant data type in casting.
  - `Length`: Specifies the length of the array.

- The following is the syntax for signature of the `CreateInstance()` method used for creating a multi-dimensional array.

Syntax

```
public static Array CreateInstance(Type elementType, int length1,
int length2)
```

- In the syntax:
  - `length1`: Specifies the row length.
  - `length2`: Specifies the column length.

- These syntax determine how the method is declared in the `Array` class.
- To create single-dimensional and multi-dimensional arrays, you must explicitly invoke the method with the appropriate parameters.

- The following code creates an array of length 5 using the `Array` class and stores the different subject names:

Snippet

```
using System;

class Subjects
{
    static void Main(string [] args)
    {
       Array objArray = Array.CreateInstance(typeof (string), 5);
       objArray.SetValue("Marketing", 0);
       objArray.SetValue("Finance", 1);
       objArray.SetValue("Human Resources", 2);
       objArray.SetValue("Information Technology", 3);
       objArray.SetValue("Business Administration", 4);
       for (int i = 0; i<= objArray.GetUpperBound(0); i++)
       {
       Console.WriteLine(objArray.GetValue(i));
       }
    }
}
```

- In the code:
  - The **Subjects** class creates an object of the `Array` class called **objArray**.
  - The `CreateInstance()` method creates a single-dimensional array and returns a reference of the `Array` class.
  - Here, the parameter of the method specifies the data type of the array.
  - The `SetValue()` method assigns the names of subjects in the **objArray**. Using the `GetValue()` method, the names of subjects are displayed in the console window.

- For manipulating an array, the `Array` class uses the following four interfaces:

ICloneable	ICollection	IList	IEnumerable
•The `Icloneable` interface belongs to the `System` namespace and contains the `Clone()` method that allows you to create an exact copy of the current object of the class.	•The `Icollection` interface belongs to the `System.Collections` namespace and contains properties that allow you to count the number of elements, check whether the elements are synchronized and if they are not, then synchronize the elements in the collection.	•The `IList` interface belongs to the `System.Collections` namespace and allows you to modify the elements defined in the array. •The interface defines three properties, `IsFixedSize`, `IsReadOnly`, and `Item`.	•The `Ienumerable` interface belongs to the `System.Collections` namespace. •This interface returns an enumerator that can be used with the `foreach` loop to iterate through a collection of elements such as an array.

Rank of an Array 1-3

- `Rank` is a read-only property that specifies the number of dimensions of an array.

Example A three-dimensional array has rank three.

- The following code demonstrates the use of the `Rank` property:

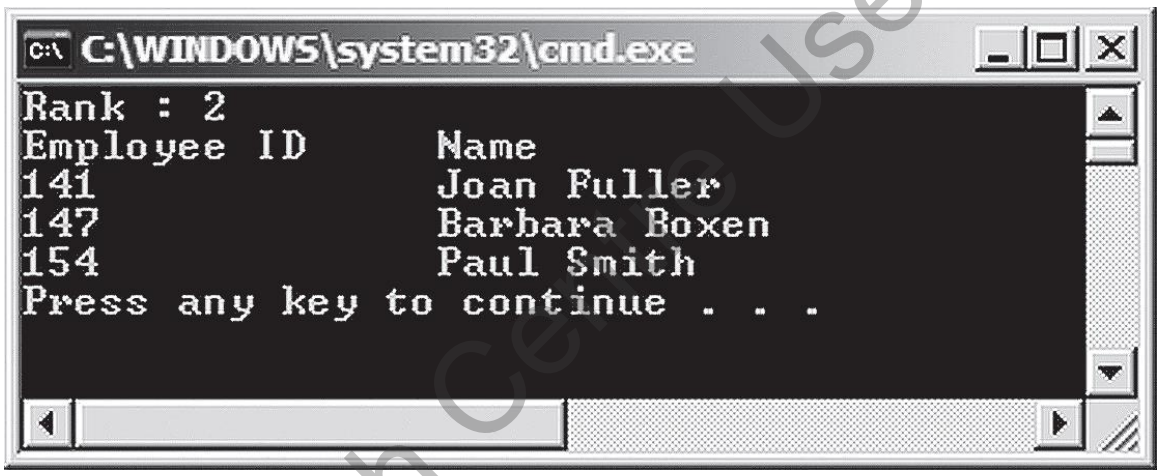
Snippet

```
using System;
class Employee
{
public static void Main()
{
    Array objEmployeeDetails = Array.CreateInstance(typeof(string), 2,3);
    objEmployeeDetails.SetValue("141", 0, 0);
    objEmployeeDetails.SetValue("147", 0, 1);
    objEmployeeDetails.SetValue("154", 0, 2);
    objEmployeeDetails.SetValue("Joan Fuller", 1, 0);
    objEmployeeDetails.SetValue("Barbara Boxen", 1, 1);
    objEmployeeDetails.SetValue("Paul Smith", 1, 2);
    Console.WriteLine("Rank : " + objEmployeeDetails.Rank);
    Console.WriteLine("Employee ID \tName");
```

```
    for (int i = 0; i< 1 ; i++)
    {
    for (int j = 0; j < 3; j++)
    {
    Console.Write(objEmployeeDetails.GetValue(i, j) + "\t\t
    ");
    Console.WriteLine(objEmployeeDetails.GetValue(i+1, j));
    }
    }
    }
}
```

- In the code:
  - The `CreateInstance()` method creates a two-dimensional array of the specified type and dimension lengths.
  - Since this array has two dimensions, its rank will be 2.
  - An instance of this class **`objEmployeeDetails`** is created and two sets of values are then inserted in the object **`objEmployeeDetails`** using the method **`SetValue()`**.
  - The values stored in the array are employee ID and the name of the employee. The `Rank` property retrieves the rank of the array which is displayed by the `WriteLine()` method.

- The following figure displays the use of `Rank` property:



Summary

- Arrays are a collection of values of the same data type.
- C# supports zero-based index feature.
- There are two types of arrays in C#: Single-dimensional and Multi-dimensional arrays.
- A single-dimensional array stores values in a single row whereas a multi-dimensional array stores values in a combination of rows and columns.
- Multi-dimensional arrays can be further classified into rectangular and jagged arrays.
- The Array class defined in the System namespace enables to create arrays easily.
- The Array class contains the CreateInstance() method, which allows you to create single and multi-dimensional arrays.