Antony Roshan

**CSHARP-2**

**Array**

* Used to store a collection of data
* Stored fixed size sequential collection of elements of same type
* Specific element is accesed by index
* **Single dimensional array**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

internal class Program

{

static void Main(string[] args)

{

int[] arr = new int[] { 7, 3, 5, 1, 9, 2 };

Console.WriteLine(arr[0]);

foreach(int item in arr)

{

Console.WriteLine(item);

}

Console.ReadLine();

}

}

}

**Output**

**7**

**7**

**3**

**5**

**1**

**9**

**2**

* **Multi dimensional array**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

internal class Program

{

static void Main(string[] args)

{

int[,] arr = new int[3,4] { {1,2,3,4 },{ 7, 3, 5, 1 },{ 9, 2, 4, 6 } };

for(int i=0;i<3;i++)

{

for(int j=0;j<4;j++)

{

Console.Write(arr[i, j] + "\t");

}

Console.WriteLine();

}

Console.ReadLine();

}

}

}

**Output**

1 2 3 4

7 3 5 1

9 2 4 6

* **Jagged array**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

internal class Program

{

static void Main(string[] args)

{

int[][] arr = new int[3][];

arr[0] = new int[3] { 1, 2, 3 };

arr[1] = new int[2] { 10, 20 };

arr[2] = new int[4] { 11, 22 ,33,44};

foreach (int[] ar in arr)

{

foreach(int item in ar)

{

Console.Write(item + "\t");

}

Console.WriteLine();

}

Console.ReadLine();

}

}

}

**Output**

1 2 3

10 20

11 22 33 44

**Methods**

* Input can be given as parameters
* Process results are returned by return keyword

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

internal class Program

{

static void SampleMethod()

{

Console.WriteLine("This is a sample method.....");

}

static void Main(string[] args)

{

Console.WriteLine("Program Started...");

SampleMethod();

Console.WriteLine("Program Ended...");

Console.ReadLine();

}

}

}

**Output**

Program Started...

This is a sample method.....

Program Ended...

**Addition of two numbers using methods**

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

internal class Program

{

static int add(int x,int y)

{

int sum = x+ y;

return sum;

}

static void Main(string[] args)

{

int num1 = 20, num2 = 10;

int result=add(num1,num2);

Console.WriteLine(result);

Console.ReadLine();

}

}

}

Methods can be called in following ways:

* Call by value
* Call by reference
* Call by output
* Call by params

**Strings**

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

internal class Program

{

static void Main(string[] args)

{

string fname, lname;

fname = "Lionel";

lname = "Messi";

//string fullname = fname +" "+ lname;

string fullname=string.Concat(fname, lname);

Console.WriteLine("fullname : "+ fullname);

//by using string constructors

char[] letters = { 'H', 'e', 'l', 'l', 'o' };

string greetings = new string(letters);

Console.WriteLine("Greetings : " + greetings);

char[] ch = greetings.ToCharArray();

//methods returnong string

string[] sarray = { "Hello", "World" };

string message = string.Join(" ", sarray);

Console.WriteLine("Message: {0}",message);

//formatting method to convert a value

DateTime waiting = new DateTime(2012, 10, 10, 17, 58, 1);

string chat = string.Format("Message sent at {0:t} on {0:D}", waiting);

Console.WriteLine("Message:" + chat);

Console.ReadLine();

}

}

}

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Security.Cryptography;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

internal class Program

{

static void Main(string[] args)

{

string str = "Hello World";

string upper = str.ToLower();

string lower = str.ToUpper();

Console.WriteLine(upper);

Console.WriteLine(lower);

Console.WriteLine(str.StartsWith("HE")); //false

Console.WriteLine(str.StartsWith("He")); //true

Console.WriteLine(str.EndsWith("ld")); //true

Console.WriteLine(str.Contains("World")); //true

Console.WriteLine(str.Equals(upper)); //false

Console.WriteLine(str.Equals(upper,StringComparison.CurrentCultureIgnoreCase)); //true

Console.WriteLine(str.IndexOf("or",StringComparison.CurrentCultureIgnoreCase));

Console.WriteLine(str.IndexOf("or",2, StringComparison.CurrentCultureIgnoreCase));

Console.WriteLine(str.Length);

Console.ReadLine();

}

}

}

**String Builder**

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Security.Cryptography;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

internal class Program

{

static void Main(string[] args)

{

string str = "Hello ";

str = str + "World";

Console.WriteLine(str);

StringBuilder sb = new StringBuilder(str); //Muttable version of string

sb.Append("World");

Console.WriteLine(str);

Console.ReadLine();

}

}

}

**Preprocessor Directives**

#define PI

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Security.Cryptography;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

internal class Program

{

static void Main(string[] args)

{

#if (PI)

Console.WriteLine("PI is defined");

#else

Console.Wrieline("PI is not defined");

#endif

Console.ReadLine();

}

}

}

**Object Oriented Programing**

* Is a programming paradigm
* Design Blueprint before implementing
* Based on the concept of objects
* Objects may contain data and functionality

**Principle concepts of OOP**

* Class
* Object
* Encapsulation
* Abstraction
* Inheritance
* Polymorphism

**Implementing Class**

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Security.Cryptography;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

class Test

{

int marks;

void CalculatePercent()

{

int percent = this.marks \* 100 / 50;

Console.WriteLine(percent);

}

static void Main(string[] args)

{

Test t1 = new Test();

t1.marks = 43;

t1.CalculatePercent();

Test t2 = new Test();

t2.marks = 38;

t2.CalculatePercent();

Console.ReadLine();

}

}

}

**Class Members**

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Security.Cryptography;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

class Test

{

int marks; //instance member

static int maxmarks = 50; //class member

void CalculatePercent()

{

int percent = this.marks \* 100 / Test.maxmarks;

Console.WriteLine(percent);

}

static void Main(string[] args)

{

Test t1 = new Test();

t1.marks = 43;

t1.CalculatePercent();

Console.ReadLine();

}

}

}

**Constructors**

* Same name as class
* Initializes the data members
* Class can have multiple constructors
* Can be classified into:
* Default Constructors
* Parameterized Constructors
* Copy Constructors
* Static Constructors

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Security.Cryptography;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

class Test

{

int marks; //instance member

static int maxmarks = 50; //class member

Test() //Default Constuctor

{

this.marks = 30;

}

Test(int marks) //Parameterised Constructor

{

this.marks=marks;

}

Test(Test t) //Copy constructor

{

this.marks = t.marks;

}

static Test() //Static Constructor

{

Test.maxmarks = 50;

}

void CalculatePercent()

{

int percent = this.marks \* 100 / Test.maxmarks;

Console.WriteLine(percent);

}

static void Main(string[] args)

{

Test t1 = new Test();

t1.CalculatePercent()

Test t2 = new Test(35);

t2.CalculatePercent();

Test t2 = new Test(t2);

t2.CalculatePercent();

Console.ReadLine();

}

}

}

**Object Initializer**

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Security.Cryptography;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

class Person

{

int age;

String name;

Char gender;

static void Main(string[] args)

{

Person p1=new Person { name = "Rahul", gender = "Male", age = 22 };

Console.ReadLine();

}

}

}

**Method Overloading**

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Security.Cryptography;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

class Person

{

public string Name { get; set; }

public string Age { get; set; }

public virtual void work()

{

Console.WriteLine("It works...");

}

}

class Employee : Person

{

public double Salary { get; set; }

public string Designation { get; set; }

public override void work()

{

Console.WriteLine("It earns...");

}

}

class Program

{

static void Main(string[] args)

{

Person p1=new Employee();

p1.work();

Console.ReadLine();

}

}

}

**Operator Overloading**

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Security.Cryptography;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

class Length

{

public Length()

{

this.feet = 0;

this.inch = 0;

}

public Length(int feet,int inch)

{

this.feet = feet;

this.inch = inch;

}

public static Length operator +(Length l1,Length L2)

{

Length l3 = new Length();

l3.feet = l1.feet + l2.feet;

l3.inch = l1.inch + L2.inch;

if(l3.inch >= 12)

{

l3.feet++;

l3.inch -= 12;

}

return l3;

}

string GetLength()

{

return string.Format("Length : {0}' {1}\"",feet,inch);

}

static void Main(string[] args)

{

Length len1= new Length(2,8);

Length len2= new Length(4,5);

Length len3 = len1 + len2;

Console.WriteLine(len1.GetLength());

Console.WriteLine(len2.GetLength());

Console.WriteLine("Total : " + len3.GetLength());

Console.ReadLine();

}

}

}

**Access Specifiers**

Can be classified into:

* Private

Cannot access outside the class

* Internal

Can access any class within the current assembly or project

* Public

Can access In external project

* Protected

Along with the base class this member can be accessed in the base class

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Security.Cryptography;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

public class MyClass1

{

internal int a; //external accessible

public int b //Accessible internally

}

class program

{

static void Main(string[] args)

{

MyClass1 m = new MyClass1();

m.a = 100;

Console.ReadLine();

}

}

}

**Properties**

* Named member in class
* Used to provide restricted access to data members
* Accessors like ‘set’ and ‘get’ are used for validations

using ConsoleApp1;

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Runtime.CompilerServices;

using System.Security.Cryptography;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

class Accounts

{

float init\_amount;

static float interest;

public float InitialAmount

{

set

{

if(value < 1000)

{

Console.WriteLine("Initial amount cannot be less than 1000");

return;

}

init\_amount = value;

}

get

{

return init\_amount;

}

}

public static float InterestRate

{

private set { interest = value; }

get { return interest; }

}

public Accounts()

{

this.InitialAmount = 10000;

}

public Accounts(float amt)

{

this.InitialAmount = amt;

}

static Accounts()

{

Accounts.InterestRate = 9.5f;

}

}

}

class program

{

static void Main(string[] args)

{

Accounts ac1=new Accounts();

ac1.InitialAmount = 500;

Console.WriteLine(ac1.InitialAmount);

Console.WriteLine(Accounts.InterestRate);

Console.ReadLine();

}

}

**Indexer**

* Allows an object to be indexed such as an array
* Instance of this class can be accessed by using the array access operator {[]}
* Use get and set accessors for defining an indexer
* Index can only be defined by **this** keyword

using System;

namespace ConsoleApp1

{

class Employee

{

public int Id { get; set; }

public string EmpName { get; set; }

public double Salary { get; set; }

}

class Department

{

public int DeptId { get; set; }

public string DeptName { get; set; }

Employee[] EmpList;

public Department()

{

DeptId = 10;

DeptName = "Sales";

EmpList = new Employee[3]

{

new Employee {Id=101, EmpName="Alex", Salary=50000},

new Employee {Id=102, EmpName="Brad", Salary=45000},

new Employee {Id=103, EmpName="Chris", Salary=40000}

};

}

public Employee GetEmployee(int id)

{

foreach (Employee emp in EmpList)

{

if (id == emp.Id)

return emp;

}

return null;

}

public Employee GetEmployee(string name)

{

foreach (Employee emp in EmpList)

{

if (name == emp.EmpName)

return emp;

}

return null;

}

public Employee this[int id]

{

get

{

foreach (Employee emp in EmpList)

{

if (id == emp.Id)

return emp;

}

return null;

}

}

public Employee this[string name]

{

get

{

foreach (Employee emp in EmpList)

{

if (name == emp.EmpName)

return emp;

}

return null;

}

}

}

class program

{

static void Main(string[] args)

{

Department dept = new Department();

Console.WriteLine(dept[101].EmpName);

Console.WriteLine(dept["Brad"].Id);

Console.ReadLine();

}

}

}

**Inheritance**

* Inheritance allows us to define a class in terms of another class
* One class(base) shares its members with another class(derived), which allows reusability
* When a derived class is lacking the required member, it takes from base class
* Base and derived classes have 'is a' relationship

using System;

using System.Data;

namespace ConsoleApp1

{

class MyClass1

{

protected int a; //only accessible in derived calss and class i which it is defined

internal int b;

public void method1()

{

Console.WriteLine("Method from MyClass1");

}

}

class MyClass2 : MyClass1

{

public void method2()

{

Console.WriteLine("Method from MyClass2");

}

}

class program

{

static void Main(string[] args)

{

MyClass1 obj1 = new MyClass1();

MyClass2 obj2 = new MyClass2();

obj2.method1();

obj2.method2();

Console.ReadLine();

}

}

}

**Data Hiding**

using System;

namespace ConsoleApp1

{

class Person

{

public string Name { get; set; }

public int Age { get; set; }

public void work()

{

Console.WriteLine("It works...");

}

}

class Employee : Person

{

public double Salary { get; set; }

public string Designation { get; set; }

public new void work() //Ignore definition provided by base class

{

Console.WriteLine("It earns....");

}

}

class program

{

static void Main(string[] args)

{

Employee emp1 = new Employee();

emp1.work();

Console.ReadLine();

}

}

}

**Method Overriding**

* Performs dynamic polymorphism
* Allows you to invoke method that belongs to different class in same hierarchy using the base class reference
* Virtual and Override keywords are used to implement method overriding

using System;

namespace ConsoleApp1

{

class Person

{

public string Name { get; set; }

public int Age { get; set; }

public virtual void work() //virtual

{

Console.WriteLine("It works...");

}

}

class Student : Person

{

public override void work() //override

{

Console.WriteLine("It studies....");

}

}

class Employee : Person

{

public double Salary { get; set; }

public string Designation { get; set; }

public override void work() //override

{

Console.WriteLine("It earns....");

}

}

class Manager : Employee

{

public sealed override void work() //cannot override further

{

Console.WriteLine("It manages a team....");

}

}

class BranchManager : Manager

{

//overrides from base class

}

class program

{

static void Main(string[] args)

{

Person p1 = new Student();

p1.work();

p1 = new Employee();

p1.work();

p1 = new BranchManager();

p1.work();

Console.ReadLine();

}

}

}