**Hello world**

using System; // System is name space

namespace CsharpConsoleApp

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Hello World");

}

}

}

**Data Types And Variable**

class Program

{

static void Main(string[] args)

{

string message = "Hello World";

int intVar = 10;

float floatVar = 10.2f;

char charVar = 'E';

bool boolVar = false;

Console.WriteLine(message);

Console.WriteLine(intVar);

Console.WriteLine(floatVar);

Console.WriteLine(charVar);

Console.WriteLine(boolVar);

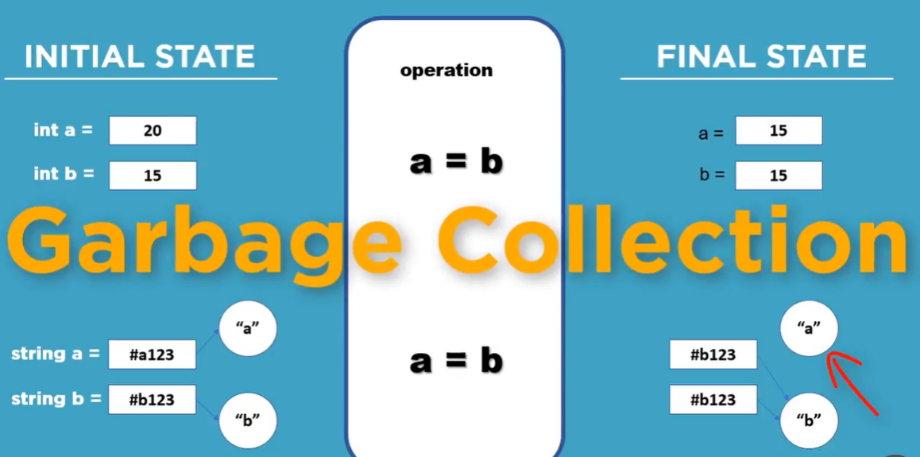
Console.ReadKey();

}

}

**Value vs Reference Type**

Example: class, objects, array, Strings, Interface etc..



**Type conversion**

1-Implicit = from smaller to larger data type

2-Explicit = Larger to small data type

// implicit

int num = 1231241234;

long newNum = num;

// explicit

double x = 3434.34;

int y = (int)x;

**Constant**

class Program

{

static void Main(string[] args)

{

// Area

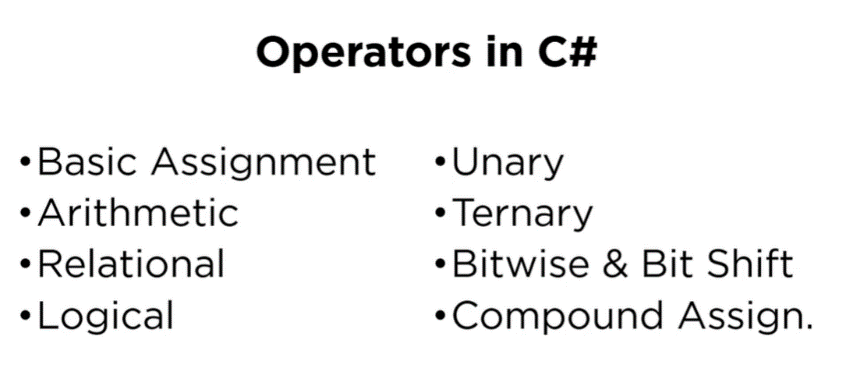
const double PI = 3.14;

PI = 100; // error cost constant can't change

}

}

Operator



**Basic assignment**

// assigning operator

int a = 10;

int b = a; //

**Arithmetic**

double firstNr = 5.4, secondNr = 4.34, result;

// add

result = firstNr + secondNr;

Console.WriteLine("{0} + {1} = {2}", firstNr, secondNr, result);

// sub

result = firstNr - secondNr;

Console.WriteLine("{0} - {1} = {2}", firstNr, secondNr, result);

// multi

result = firstNr \* secondNr;

Console.WriteLine("{0} x {1} = {2}", firstNr, secondNr, result);

// div

result = firstNr / secondNr;

Console.WriteLine("{0} / {1} = {2}", firstNr, secondNr, result);

// modul

result = firstNr % secondNr;

Console.WriteLine("{0} % {1} = {2}", firstNr, secondNr, result);

**Relation Operator ==, > ,< ,>= ,<= !=**

double firstNr = 5.4, secondNr = 4.34;

bool result;

result = firstNr == secondNr;

Console.WriteLine("{0} == {1} = {2}", firstNr, secondNr, result);

result = firstNr > secondNr;

Console.WriteLine("{0} > {1} = {2}", firstNr, secondNr, result);

result = firstNr < secondNr;

Console.WriteLine("{0} < {1} = {2}", firstNr, secondNr, result);

result = firstNr >= secondNr;

Console.WriteLine("{0} >= {1} = {2}", firstNr, secondNr, result);

result = firstNr <= secondNr;

Console.WriteLine("{0} <= {1} = {2}", firstNr, secondNr, result);

result = firstNr != secondNr;

Console.WriteLine("{0} != {1} = {2}", firstNr, secondNr, result);

**Logical Operator || And &&**

double firstNr = 10, secondNr = 4.34;

bool result;

result = (firstNr == secondNr) || (firstNr > 5);

Console.WriteLine(result);

result = (firstNr == secondNr) && (firstNr > 5);

Console.WriteLine(result);

**Unary Operator +var, -var ,++var, --var, !var**

int number = 10, result;

bool flag = true;

result = +number;

Console.WriteLine("+number = "+result);

result = -number;

Console.WriteLine("-number = "+result);

result = ++number;

Console.WriteLine("++number = " + result);

result = --number;

Console.WriteLine("--number = "+result);

Console.WriteLine("!flag = " + !flag);

**Ternary Operator variable = condition ? expression1 : expression2;**

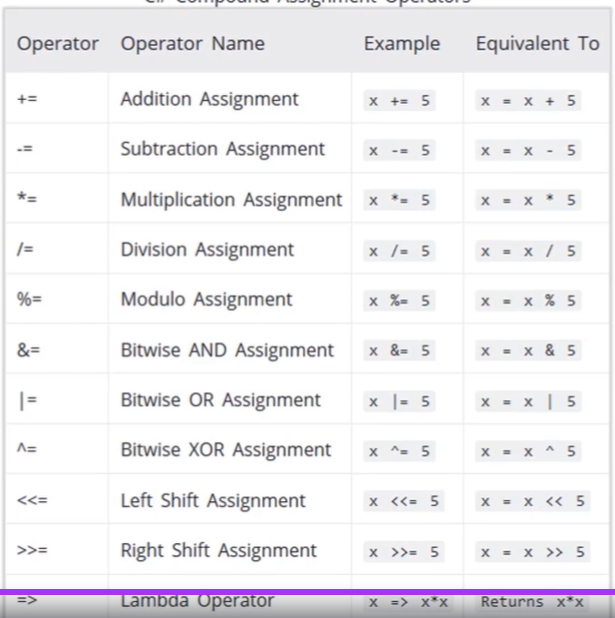
int number = 10;

string result;

result = (number > 0) ? "Greater" : "Not Greater";

Console.WriteLine("{0} is {1}", number, result);

**Compound operator**



int number = 10;

number += 5;

Console.WriteLine(number);

number &= 5;

Console.WriteLine(number); //number = 5

number ^= 5;

Console.WriteLine(number); //number = 0

**Decision Statement**

int number = 0;

if(number > 0)

{

Console.WriteLine("number is greater");

}

else

{

Console.WriteLine("number is positive");

}

switch (number)

{

case 0: Console.WriteLine("number is greater"); break;

case 1: Console.WriteLine("number is positive"); break;

default: Console.WriteLine("number is default"); break;

}

**Loop (for, while, do-while)**

for(int i = 1; i< 11; i++)

{

Console.WriteLine($"{i} - Hello!");

}

int j = 1;

while(j< 11)

{

Console.WriteLine($"{j} - Hello!");

j++;

}

j = 0;

do

{

Console.WriteLine($"{j} - Hello!");

j++;

} while (j < 11);

**Access Modifiers**

**public, private, protected, internal, protected internal**

**public:** accessed from anywhere within the application.

**private:** accessed from only within the class.

**protected:** can only be accessed from the same class and its derived classes (child).

**internal**: can be accessed only within the same assembly.

**Protected internal :** can be accessed from the same assembly and the derived class of the containing class from any other assembly

**Function**

internal class Program

{

static void Main(string[] args)

{

double firstSum = addNumbers(12.5, 12.5);

Console.WriteLine($"First sum = {firstSum}");

hello();

}

public static double addNumbers(double firstNumber, double secondNumber)

{

return firstNumber + secondNumber;

}

public static void hello()

{

Console.WriteLine("Hello");

}

}

**Structure**

Student.cs

namespace ConsoleApp

{

public struct Student

{

public string FirstName { get; set; }

public string LastName { get; set; }

public int Grade { get; set; }

//default constructor

public Student(string fname, string lname, int grade)

{

this.FirstName = fname;

this.LastName = lname;

this.Grade = grade;

}

public void ShowStudent()

{

Console.WriteLine($"{FirstName} {LastName} {Grade}");

}

}

}

Program.cs

internal class Program

{

static void Main(string[] args)

{

Student student = new Student();

student.FirstName = "Test";

student.LastName= "Test";

student.Grade= 1;

Student student1 = new Student();

student1.FirstName = "Test2";

student1.LastName = "Test2";

student1.Grade = 3;

Console.WriteLine($"{student.FirstName} {student.LastName} {student.Grade}");

Console.WriteLine($"{student1.FirstName} {student1.LastName} {student1.Grade}");

Student student2 = new Student("test3", "test3", 3);

student2.ShowStudent();

}

}

**Enum**

internal class Program

{

static void Main(string[] args)

{

Console.WriteLine(WeekDays.Friday);

Console.WriteLine((int)WeekDays.Thursday);

Console.WriteLine("GetName()");

Console.WriteLine(Enum.GetName(typeof(WeekDays), 5));

Console.WriteLine("GetNames()");

foreach(string str in Enum.GetNames(typeof(WeekDays)))

{

Console.WriteLine(str);

}

Console.WriteLine("TryParse()");

WeekDays wdEnum;

Enum.TryParse<WeekDays>("1", out wdEnum);

Console.WriteLine(wdEnum);

}

enum WeekDays

{

Monday = 1,

Tuesday,

Wednesday,

Thursday,

Friday,

Saturday,

Sunday

}

}

**StringBuilder**

StringBuilder sb = new StringBuilder("This is string builder");

var sbNew = sb;

sb.Append("This is additional");

Console.WriteLine(ReferenceEquals(sbNew, sb));

StringBuilder sbBuilder = new StringBuilder();

sbBuilder.Append("I have");

sbBuilder.AppendFormat("{0:C}", 25);

sbBuilder.Insert(0, "Join : ");

sbBuilder.Remove(1, 2);

sbBuilder.Replace("have", "had");

Console.WriteLine(sbBuilder);

**Array**

int[] numbers = new int[5];

numbers[0] = 10;

numbers[1] = 20;

numbers[2] = 4;

numbers[3] = 40;

Array.Sort(numbers);

Array.Reverse(numbers);

foreach(int i in numbers)

{

Console.WriteLine(i);

}

string[] studentNames = new string[10];

double[] studentResult = new double[4]

{

5, 6.5, 10, 11

};

**Class**



Circle.cs

using System;

namespace CsharpConsoleApp

{

public class Circle

{

// field

private const double PI = 3.14;

// property

public double Radius { get; set; }

public Circle()

{

Console.WriteLine("Circle() Created");

}

public Circle(double Radius)

{

this.Radius = Radius;

}

// method

public double GetArea() => PI \* Math.Pow(Radius, 2);

public double GetPerimeter() => 2 \* PI \* Radius;

}

}

Program.cs

static void Main(string[] args)

{

Circle circle = new Circle();

circle.Radius = 5;

double c1 = circle.GetArea();

Console.WriteLine(c1);

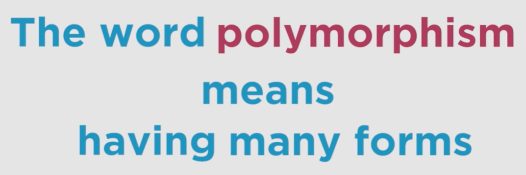
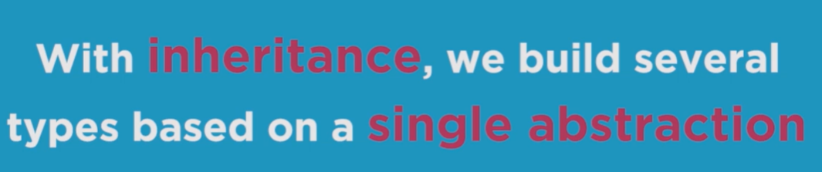
Circle circle1 = new Circle(5);

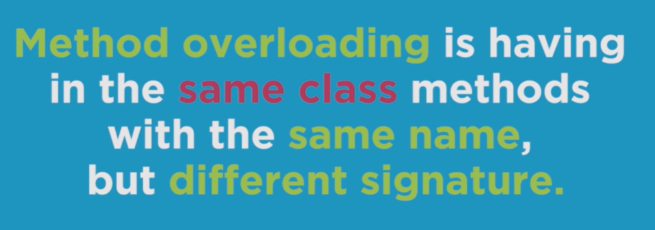
double c2 = circle.GetArea();

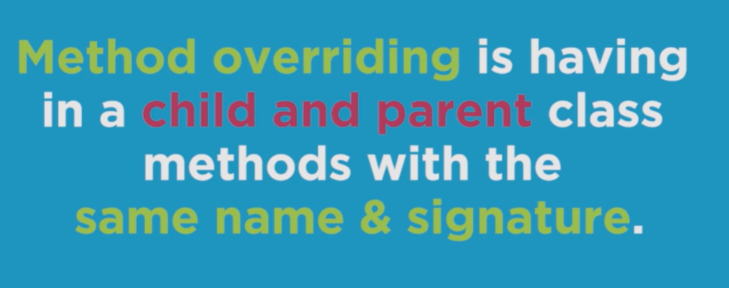
Console.WriteLine(c2);

}

**Inheritance and plymorphishm**







Student.cs

namespace CsharpConsoleApp

{

public class Student:Person

{

public int Grade { get; set; }

public override string GetDetail() => $"{FullName} of age {Age} grade = {Grade}";

}

}

Professor.cs

namespace CsharpConsoleApp

{

public class Professor:Person

{

public string Subject { get; set; }

public override string GetDetail() => $"{FullName} of age {Age} Subject = {Subject}";

}

}

Person.cs

namespace CsharpConsoleApp

{

public class Person

{

public string FullName { get; set; }

public int Age { get; set; }

public virtual string GetDetail() => $"{FullName} of age {Age}";

//overloading

public string GetDetail(double height) => $"{FullName} of age {Age}";

}

}

Program.cs

class Program

{

static void Main(string[] args)

{

Person person = new Person() { Age = 15, FullName="Join Son"};

Console.WriteLine(person.GetDetail());

Professor processor = new Professor() { Age = 45, FullName = "Lixa", Subject = "Math" };

Console.WriteLine(processor.GetDetail());

Student student = new Student() { Age = 20, FullName = "Eyjo", Grade = 2 };

Console.WriteLine(student.GetDetail());

}

}

**Interface**



**Iperson.cs**

namespace CsharpConsoleApp

{

public interface IPerson

{

string FullName { get; set; }

int Age { get; set; }

string GetDetail();

string GetDetail(double height);

}

public interface ITest

{

void Test();

}

}

**Person.cs**

namespace CsharpConsoleApp

{

public class Person : IPerson

{

public string FullName { get ; set; }

public int Age { get; set; }

public string GetDetail() => $"{FullName} of age {Age}";

public string GetDetail(double height) => $"{FullName} of age {Age}";

}

}

**Professor.cs (Can implement more interface)**

namespace CsharpConsoleApp

{

public class Professor : IPerson, ITest

{

public string Subject { get; set; }

public string FullName { get => throw new NotImplementedException(); set => throw new NotImplementedException(); }

public int Age { get => throw new NotImplementedException(); set => throw new NotImplementedException(); }

public string GetDetail()

{

throw new NotImplementedException();

}

public string GetDetail(double height)

{

throw new NotImplementedException();

}

public void Test()

{

throw new NotImplementedException();

}

}

}

**Student.cs**

namespace CsharpConsoleApp

{

public class Student : IPerson

{

public string FullName { get => throw new NotImplementedException(); set => throw new NotImplementedException(); }

public int Age { get => throw new NotImplementedException(); set => throw new NotImplementedException(); }

public string GetDetail()

{

throw new NotImplementedException();

}

public string GetDetail(double height)

{

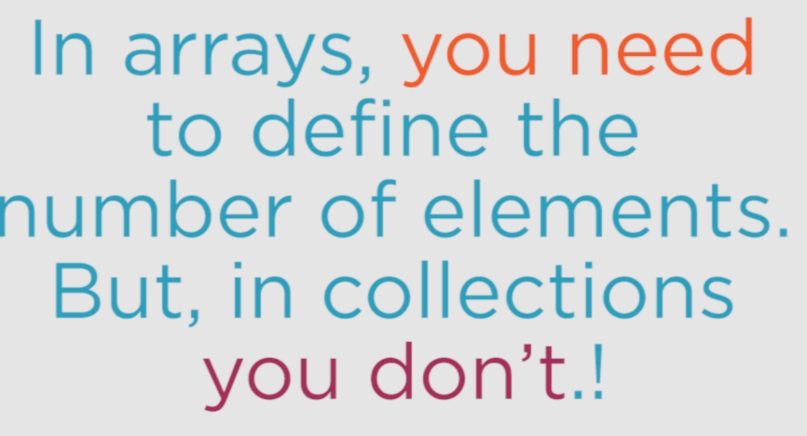
throw new NotImplementedException();

}

}

}

**Collection**







**Array Lists**

You don’t need to specify the size of arraylist

ArrayList myValue = new ArrayList();

myValue.Add(10);

myValue.Add("this is a string");

myValue.Add(45.9);

// append value to end arraylist

myValue.AddRange(new ArrayList() { "John", 24, 78, 78 });

myValue.Insert(1, "new value");

myValue.Remove(45.9);

myValue.RemoveRange(0, 2);

myValue.RemoveAt(0);

foreach(var value in myValue)

{

Console.WriteLine(value);

}

//================

ArrayList myValue = new ArrayList();

myValue.Add(10);

myValue.Add(45);

// append value to end arraylist

myValue.AddRange(new ArrayList() { 24, 78, 78 });

// if sort we need to same data type of value

myValue.Sort();

myValue.Reverse();

Console.WriteLine(myValue.Contains(45));

myValue.Clear();

foreach (var value in myValue)

{

Console.WriteLine(value);

}

**SortedList**

SortedList list = new SortedList();

list.Add(3, "Number one");

list.Add(1, "Number two");

list.Add(2, "Number three");

// key will be same data type

string value = (string)list[1];

Console.WriteLine(value);

for(int i = 0; i< list.Count; i++)

{

Console.WriteLine($"Key: {list.GetKey(i)}, value: {list.GetByIndex(i)}");

}

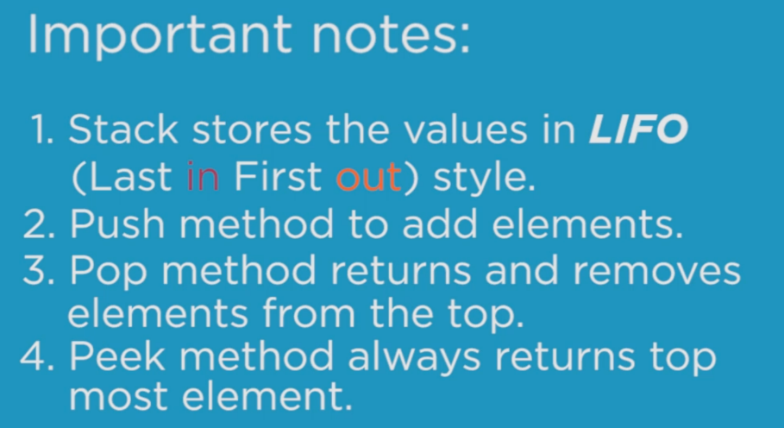
foreach(DictionaryEntry element in list)

{

Console.WriteLine($"Key: {element.Key}, value: {element.Value}");

}

**Stack**



Stack myStack = new Stack();

myStack.Push("start stack");

myStack.Push(23);

myStack.Push(4.3);

myStack.Push(null);

myStack.Push("end stack");

Console.WriteLine(myStack.Count);

Console.WriteLine(myStack.Peek());

// remove item

myStack.Pop();

Console.WriteLine(myStack.Contains(3));

foreach (var stackItem in myStack)

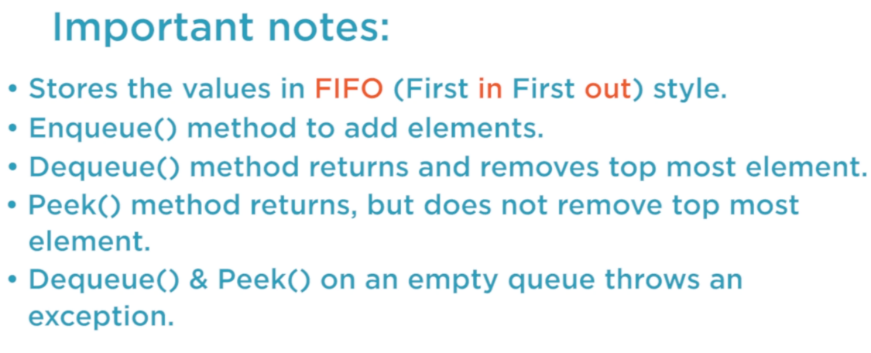
{

Console.WriteLine(stackItem);

}

myStack.Clear();

**Queue**



Queue myQueue = new Queue();

myQueue.Enqueue(1);

myQueue.Enqueue(2);

myQueue.Enqueue(3);

myQueue.Enqueue("this my string");

//myQueue.Enqueue(null);

Console.WriteLine($"Number of element = {myQueue.Count}");

foreach(var queue in myQueue)

{

Console.WriteLine(queue);

}

Console.WriteLine($"First of Elements = {myQueue.Peek()}");

Console.WriteLine($"Search of Elements = {myQueue.Contains(3)}");

while (myQueue.Count> 0)

{

Console.WriteLine(myQueue.Dequeue());

}

Console.WriteLine($"Number of Elements = {myQueue.Count}");

**Hashtable**

Key cannot be null but value can

Hashtable myHashTable = new Hashtable()

{

{3, "Three" },

{4, "Four" },

{5, "Five" }

};

myHashTable.Add(6, "One");

myHashTable.Add("7", "Two");

//====================

Dictionary<int, string> myDictionary = new Dictionary<int, string>();

myDictionary.Add(1, "First element");

myDictionary.Add(2, "Second element");

Hashtable myHashTable1 = new Hashtable(myDictionary);

string value = (string)myHashTable1[1];

Console.WriteLine($"Eelement is {value}");

Console.WriteLine($"Search {myHashTable1.ContainsValue("First element")}");

myHashTable1.Remove(1);

myHashTable1.Clear();

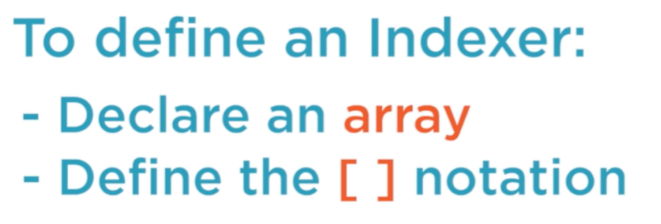
foreach (DictionaryEntry item in myHashTable1)

{

Console.WriteLine($"Key: {item.Key}, value: {item.Value}");

}

**Indexer**



class Program

{

static void Main(string[] args)

{

SampleConllection sampleConllection = new SampleConllection();

sampleConllection[0] = 25;

Console.WriteLine(sampleConllection[0]);

}

}

public class SampleConllection

{

private int[] array = new int[60];

public int this[int i]

{

get { return array[i]; }

set { array[i] = value; }

}

}

**Exception**



try

{

int x = 30;

Console.WriteLine($"12/0 = {x / 0}");

}

catch (Exception e)

{

Console.WriteLine(e);

throw;

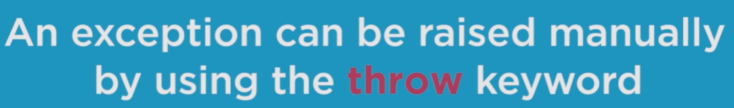
}

finally

{

Console.WriteLine($"finaly block");

}



int positiveNum = -10;

if (positiveNum <= 0) throw new ArgumentException("The number is not positive!");

//=============

class Program

{

static void Main(string[] args)

{

Student st = new Student();

try

{

st.Age = -10;

InvalidStudentAgeException(st.Age);

}catch(InvalidStudentAgeException e)

{

Console.WriteLine(e.ToString());

throw;

}

}

private static void InvalidStudentAgeException(int age)

{

if (age < 18) throw new InvalidStudentAgeException(age);

}

}

public class Student

{

public int StudentID { get; set; }

public int Age { get; set; }

}

public class InvalidStudentAgeException : Exception

{

public InvalidStudentAgeException()

{

}

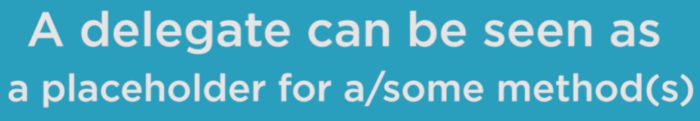
public InvalidStudentAgeException(int age) : base($"The number {age} is not valid")

{

}

}

**Delegate**



namespace CsharpConsoleApp

{

public delegate void StringDelegate(string text);

class Program

{

static void Main(string[] args)

{

StringDelegate stringDelegate = ToUpperCase;

stringDelegate("This is lowercase");

stringDelegate.Invoke("This other is lowercase");

stringDelegate = ToLowerCase;

stringDelegate("This is ToUpper");

}

static void ToUpperCase(string text) => Console.WriteLine(text.ToUpper());

static void ToLowerCase(string text) => Console.WriteLine(text.ToLower());

}

}

namespace CsharpConsoleApp

{

public delegate void StringDelegate(string text);

class Program

{

static void Main(string[] args)

{

StringDelegate stringDelegate = ToUpperCase;

WriteOutput("Hello World", stringDelegate);

}

static void ToUpperCase(string text) => Console.WriteLine(text.ToUpper());

static void ToLowerCase(string text) => Console.WriteLine(text.ToLower());

static void WriteOutput(string text, StringDelegate stringDelegate)

{

Console.WriteLine($"Before: {text}");

stringDelegate(text);

}

}

}

Event