Lesson 19 Method parameters

Out Parametes: use when you want a method to return more than one value

Do(int a, int b, out add, out times)

{}

Parameter arryas: 允许takes a variables number of arguments. 可以是array, list of arguments or no argument,Params keyword永远是最后一个用的

比如：

Public void paramethod (params int [] Numbers)

{ console.writeline(Numbers.Length )};

在main里可以叫 paramethod(1,2,3,4,5); 输入时5；

Paramethod (); 输出是0

Int [] numbers = new int[3];

Paramethod(numbers); 输出是3；

Lesson 18 Namespace:

Why Namespace: Namespaces are used to organize ur program. They aslo provide assistance in avoiding name clashes.

可以这么用

using tsk = System.Threading;

Lesson 19 static and instance member

Static member are invoked using class name where as instance member are invoked using instances of the class. An Instance member belongs to specific instance of a class.

不是static 就是instance的

Static variable inside class 不能用this

比如 static float \_PI = 3.14 不能用this.\_PI, 必须用Circle.PI 必须用class的名字去access

Static constructor 会run before public constructor 当你call 它的static field; static constructor cannot use access modifier

Why do we need static constructor to 给static field 赋值

Static float \_PI;

Static Circle () {

Circle.PI = 3.14F;

}

Lesson 21:

1. C# 只支持一个class 的inheritance
2. C# 支持多个interface inheritance
3. Child class is a specialization of base class
4. Base classes are automatically instantiated before derived class
5. Base class non default constructor 需要通过 initialize list 初始化 Base

class ParentClass {

public ParentClass(string a) { }

}

class childClass : ParentClass

{

public childClass(string input): base(input){}

}

Lesson 22 Method Hiding

如果child class 和 base class 有一样的function，当initialize 子class，并且call那个 function， 会hidden base class method(function)

假如真的想叫base class 的method,

1. 可以用cast

((parent)P).get\_name();

1. 可以用polymorphism

ParentClass P = new ChildClass();

PC.get\_name(); // 可以叫的原因: 不是virtual

用new keyword: tell compiler 故意hide base class method

class Class1

{

public string Name = "Bob";

public int age = 10;

public void showMessage(string Message) {

System.Windows.Forms.MessageBox.Show(Message);

}

}

class MysecondClass : Class1 {

public new void ShowMessage(string message) // override parent的class function

{

System.Windows.Forms.MessageBox.Show(message, "title");

}

}

}

base.get\_name(); 如果base 和child class 都有get\_name()

lesson 23: Polymorphism： allow u to invoke derived class methods through a bass class reference during runtime

override 在child function 前面， virtual 在base class前面

public virtual void showMessage(string Message)

public override void showMessage(string message)

virtual override 用 polymorphism 会call **derived** class method (Virtual)

virtual new 用 polymorphism 会call **Base** class method (Hidden)

Lesson 25: Method Overloading

Function overloading and Method Overloading terms 是一样的

Method loading 是允许class 有多个一样名字的function，但是Different signature(传入function 的参数不一样)，因此function can be overloaded base on number, type(int, float) and Kind(Value, Ref, Out) of parameters

Cannot overload method base on return type, params keyword

Lesson 26, 27 Properties

如果假如一个class，规定int 不能设置为0，name 不能 null. 比如read only

A property with both get and set is Read/Write Property

A property with only get accessor is a Read only Property

A property with only set accessor is a Write Only Property

public class Student {

private int \_id;

public int Id {

get { return this.\_id; }

set {

if (value <= 100)

throw new Exception("student ID cannot be negative");

else \_id = value;

}

}

If there is no additional logic in the property accessors, then we can make use of auto implemented properties

public int Id {

get;

set; // tell compiler to creat a private field to read and write

}

public string Name { get => \_name; set => \_name = value; }

Lesson 28 Struct

Struct classes can have

1. Private fields, 2. Public proeperties 3. Constructors 4. Methods

public struct CustomerC {

private int \_id;

private string \_name;

public int ID {

set;

get;

}

public string Name { get => \_name; set => \_name = value; }

public void Print() { }

}

CustomerC c = new CustomerC();

Lesson 29 Difference between classes and struct:

1. A struct is a value type where as class is reference type
2. Struct is stored on Stack whereas classes are stored in Heap
3. Value types(Stack) hold their value in memory where they are declared but reference types(Class) hold a reference to an object in memory.
4. Value Types are destroyed immediately after the scope is lost whereas for reference types only the reference variable is destroyed after the scope is lost. The object is later destroyed by garbage collector.
5. When you copy a struct into another struct, new copy of that struct gets created and modification on one struct will not affect the values contained by the other struct, 当struct 复制了，新的struct 值得更改，不会改变老的struct

When copy a class into another class, we only get a copy of the reference variable(只是copy 这个reference 的变量，Both reference varaibles point to the same object on the heap) 因此 operation on one variable will affect the values contained by the other reference variable

复制struct，更改新的，不会变老的 (Value types)

复制 class， 更改老的，会改新的，（reference types）

1. Struct 不能有destructor，class 这个可以有
2. Struct 不能inherit from another class, 但是class 可以， Struct and class 都可以inherit from an interface
3. Class can have parameter less constructor but Struct cannot have that

public class student {

public student ();

}

public Struct student {

public student (); // 不行 报错

}

* A class or a struct cannot inherit from another struct

Keyword Sealed : 就是不能用于继承，不能作为别的class 的base class

public sealed class student {

public student ();

}

Lesson 30 Introduction to Interfaces

Interfaces using interface keyword, Just Like classes interfaces aslo contains properties methods, delegates, events. But only declarations and no implementations.  
1. 只能定于， 不能有implementation

Interface members are public by default and they don’t allow explicit access modifier.

1. 是public by default, 不能有access modifier(public, protected, private)

相比而言 class 的access modifier 是private by default

1. 不能有field， 不能再interface 中定义 int ID;
2. It is a compile time error to provide implementations for any interface member

是compile time 获得implantation 对于interface

1. Interfaces cannot contain fields
2. 对于class 和struct 想要inherit from interface 必须有implementation for all interface members 否则不会compile
3. 一个class 或者Struct 可以一次inherit 多个 interfaces at the same time, but a class cannot inherit from more than one class at the same time
4. Interfaces can inherit from other interfaces. A calss that inherit 这个interface 必须provide 所有的implementation for all interface members in the entire interface inheritance chain.
5. 不能建立一个instance of interface but an interface variable can point to a derived class object

（interface 都没有implementation，怎么instantiate object）

但是这样可以 Icustomer i = new New\_Cust();（polymorphism）

interface Icustomer { //Common convention to name it starting with I

// int id; 错误 不能有field，

void Print(); //不能有这个method 的定义， 而且不能有public， private 的值，public by default

}

interface I2 {

void I2Method();

}

public class New\_Cust : Icustomer, I2 { // class has to provide interface implementation

public void I2Method()

{}

public void Print() {

} // 需要access modifer, 必须signature 一样

}

Lesson 31 Explicit Interfaces Implementation:

假如一个class 继承两个interface， 两个interface 都有一样的method 需要casting

interface I1 {

void Print(); //不能有这个method 的定义， 而且不能有public， private 的值，public by default

}

interface I2 {

void Print();

}

public class New\_Cust : I1, I2 { // class has to provide interface implementation

public void Print() {

} // 需要access modifer, 必须signature 一样

}

New\_Cust a = new New\_Cust();

((I1)a).Print();

((I2)a).Print();

// 如果explicit inherit interface,不能有access modifier

public class New\_Cust : I1, I2 { // class has to provide interface implementation

void I1.Print() { // 如果explicit inherit interface,不能有access modifier

}

void I2.Print()

{ // 如果explicit inherit interface,不能有access modifier

}

}

如果explicit implement interface member 比如用casting call interface method

The interface member cannot be access thru class reference variable but only thru the interface reference variable

New\_Cust a = new New\_Cust();

((I1)a).Print();

((I2)a).Print();

而不行 a.Print() 会报错，

但是假如用polymorphism 可以直接用，不用casting

I1 i1 = new New\_Cust();

i1.Print();

可以make 一个interface implementation to be default 就可以access default method by class instance

public class New\_Cust : I1, I2 { // class has to provide interface implementation

public void Print() { // 如果explicit inherit interface,不能有access modifier

}

void I2.Print()

{ // 如果explicit inherit interface,不能有access modifier

}

}

Lesson 32 Abstract Class

1. Abstract class cannot be sealed
2. Abstract class may contain abstract members (methods, properties, indexer, events) but not mandatory
3. A non-abstract class derived from an abstract class must provide implementations for all inherited abstract members

如果一个class， 继承abstract class ,必须提供所有abstract 的implementation，但是假如不想提供implementation的话，就必须marked as abstract

Abstract class cannot be sealed, sealed cannot be abstract class

子class 提供implementation 必须写override

public abstract class Customer11

{

public abstract void Print();

void Print1() { } // abstract class can provide methods, properties, indexer, and events

}

public class Pro1 : Customer11

{

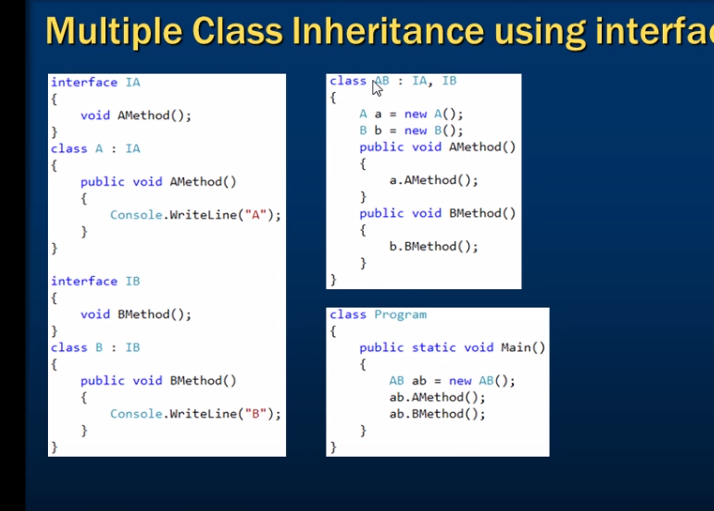
public override void Print() { }

}

Lesson 33 Abstract classes vs Interfaces

1. Abstract classes can have implementation for some members but the interface can’t have implementation for any of its member.
2. Interface cannot have fields where as an abstract class can have fields e.g public int id;
3. Interface 只能继承interface， 不能继承from abstract class where as an abstract class can inherit from another abstract class or another interface
4. A class can inherit from multiple interfaces at the same time where as a class cannot inherit from multiple classes at the same time 可以继承多个Interface 一次，但不能一次继承多个abstract class
5. Abstract class members can have access modifier whereas interface members 不能有access modifier. Abstract class 可以有public， private ，但是interface 没有default 是public

Lesson 34 Multiple class inheritance using interface



Lesson 36 Delegate:

Delegate is reference type:

Lesson 74 Dictionary in C#

Dictionary class is present in using System.Collections.Generic;

1. 初始化Dictionary：

Dictionary<int, customer> dictionaryCustomers = new Dictionary<int, customer>();

1. 往dictionary 里面加元素

dictionaryCustomers.Add(5,"12312");

1. 判断dictionary 的key是否存在(ContainsKey)或者value存在(ContainsValue), 返回值是boolean

if (dictionaryCustomers.ContainsKey(5)) {

}

if (dictionaryCustomers.ContainsValue("123")) {

}

1. Loop through dictionary
2. Loop through all(key, values) keyValuePair, var

Foreach(keyValuePair<int,customer> a in Dictionary){

Console.Writeline(a.Key, a.Value) ; //return key, value

}

Foreach(var a in Dictionary){ 用var 也可以

Console.Writeline(a.Key, a.Value) ; //return key, value

}

1. Loop through key Dictionary.Keys

Foreach(int a in Dictionary.Keys){

Console.Writeline(a) ; //a 是key

}

1. Loop through value Dictionary.Values

Foreach(int a in Dictionary.Values){

Console.Writeline(a) ; //a 是key

}

1. TryGetValue 是假如有key的话，返回true，没有找的值，返回false

参数第一个是key 值，第二个是参数值，必须是lvalue，不能是rvalue

把第二个parameter设置成找到的key对应的value，如果没有的话，设置第二个带进的参数为null

string output;

dictionaryCustomers.TryGetValue(101, out output);

1. Count() 返回有几个key

dictionaryCustomers.Count();

1. Count 数有几个value, key 的值满足条件的

int a = dictionaryCustomers.Count(kvp => kvp.Value.Salary > 4000);

kvp: key value pair

int a = dictionaryCustomers.Count(kvp => kvp.Key > 4000);

1. Remove /clear

Remove 是remove 一个key 如果不存在这个key，不会报错，nothing happen,

dictionaryCustomers.Remove(110); remove key 是110的值

clear 是清楚所有的key

dictionaryCustomers.Clear();

1. Array 变成Dictionary / List 成 Dictionary

Cutomer[] customers = new Customer [3];

Customer[0] = customer0;

Customer[1] = customer1;

Customer[2] = customer2;

Dictionary<int,Customer> dict = Customers.ToDictioanary(cust = > cust.ID, cust=> cust);

Key 是 ID， value是customer itself

Cust 代表foreach loop, For each in Customer

List<Cutomer> customer List<Customer>();

Customer.add(customer0);

Customer.add(customer1);

Customer.add(customer2);

Dictionary<int, customer> dictionaryCustomers = new Dictionary<int, customer>(); // want key to be customer ID, value

dictionaryCustomers.Add(customer1.ID, customer1);

dictionaryCustomers.Add(customer2.ID, customer2);

Customer cs119 = dictionaryCustomer[119]; //key 是119的

Foreach(keyValuePair<int,customer> a in Dictionary){

Console.Writeline(a.Key, a.Value) ; //return key, value

}

Foreach(var a in Dictionary){ 用var 也可以

Console.Writeline(a.Key, a.Value) ; //return key, value

}

Loop through key

Foreach(int a in Dictionary.Keys){

Console.Writeline(a) ; //a 是key

}

Foreach(customer a in Dictionary.Values){

Console.Writeline(a.Salary) ; //a 是values

}

判断key 是否存在

If( Dictionary.Containskey(ID) ){

}

Lesson 74, 75 List 可以grow with size, 但是array 不能grow by size

List 变成dictionary， 必须说清谁是key

1. Initialize:

List<Customer> l = new List<Customer>(2); 2是capacity

2. 插入元素

l.Add(c1);

l.Add(c2);

l.Add(c3);

3. 循环list

foreach (Customer c in l) {

}

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

}

4. list 的size： l.Count()

比如initialize 一个list List<Customer> l = new List<Customer>(2);

这是可以加class 的子class

public class SubCustomer : Customer {

}

SubCustomer sc = new SubCustomer();

l.add(sc); no error

5. Insert:

L.Insert(0,c3); 第一个parameter是位置，第二个参数是插入的变量

6. 获取特定元素的index, IndexOf

Customer.IndexOf(c3); //返回第一个是这个index的

Customer. IndexOf (c3, 2); // 第二个参数是开始搜索的位置, 从index2，开始

Customer. IndexOf (c3, 2, 2); // 第三个参数是从第二参数起，搜索的元素的个数，比如从第二个参数起，搜索连续list中两个元素，如果有，返回index，如果没有，返回-1

7

List 的function：

Contains(): 检查是否含有这个元素，如果有，返回true，如果没有，返回false

Exists() 检查元素是否在这个list base on condition

Find() 检查元素，matches the conditions defined by specified lambda expression and 返回第一个match 的在list 中 返回值是list 中的元素的type

FindLast() 检查元素满足lambda function，返回最后一个match 条件的

FindAll() 检查元素所有满足lambda function的 。 返回值一个list , list 中元素与原list 中元素的类型一样

FindIndex () : returns the index of the first item that 满足lambda function的条件的，可以有两个overload parameter 让我们定义条件。 第一个参数是起始点，第二个参数是个数，第三个参数是lambda function

FindLastIndex(): 返回list of last Item that 满足lambda function的，可以有两个overload 的parameter: 第一个参数是起始点，第二个参数是个数，第三个参数是lambda function

ToList() : 把array 变成list

ToArray(): 把list 变 array

ToDictionary(): 把List 变成dictionary

Contains:

if (l.Contains(c3)) { };

Exists: 检查Name 是以 R开始的

if (l.Exists(cust => cust.Name.StartsWith("R"))) { };

Find:

Customer c11 = l.Find(cust => cust.Salary >= 5000);

找到salary 大于5000的 第一个值

FindLast： 找到salary 大于5000的最后一个值

Customer c11 = l.FindLast(cust => cust.Salary >= 5000);

FindAll:

List<Customer> c11 = l.FindAll(cust => cust.Salary >= 5000);

FindIndex / FindLastIndex

int id = l.FindIndex(cust => cust.Salary > 5000);

从第一个元素开始，找两个元素，满足这个条件的

int id = l.FindIndex(1, 2, cust => cust.Salary > 5000);

int id = l.FindLastIndex(1, 2, cust => cust.Salary > 5000);

array 变成 list： Customer[] customerArray = new Customer[3];

customerArray[0] = c1;

customerArray[1] = c2;

customerArray[2] = c3;

List<Customer> lc = customerArray.ToList();

List 变成 array：Customer[] c33 = l.ToArray();

List 变成dictionary，必须说谁是key，

Dictionary<int,Customer>d1 = l.ToDictionary(Cust => Cust.ID );

Lesson 76 Working with generic list class and ranges in C#

1. AddRange() : 相当于extend 允许你一次加一个list，在end of arrya(), Add()允许加一个item at a tiem to the ned of the list
2. GetRange() ， 得到一个相当于subString 的list

Expect two 2 parameters, the start index in the list and the number of elements

1. InsertRange(): 把一个list 插进一个list
2. RemoveRange(): remove掉所有从第一个参数起，个数为第二个参数的元素的，Expect two 2 parameters, the start index in the list and the number of elements
3. RemoveAt()； 参数是index，remove到参数是第一个element 在list 中
4. RemoveALL(): 删除所有元素 满足lambda function的

Add Range:

List<Customer> l\_new = new List<Customer>(2);

l.AddRange(l\_new);

GetRange():

List<Customer> lll = l.GetRange(3, 2); 得到一个list，是l的起始位置index为3，然后个数是2的新list

InsertRange():

l.InsertRange(0, l\_new); 插入l\_new 到 l，位置是0

RemoveAll():

l.RemoveAll(x=>x.Type =+ “Retail”)

RemoveRange():

l.Remove(3,2); 从元素开始，remove掉两个元素

Lesson 77, 78 sort a list of simple types

1. Sorting a list of simple types like int, string ,char is straightforward

List<int> numebers = new List<int> { 1, 3, 5, 7, 8 };

numebers.Sort();

1. Reverse the list: numebers.Reverse()
2. Sort the class type list (complex Type): 需要提供IComparable interface

The complex type has to implement Icomparable interface and provide implementation for CompareTo() method. CompareTo() method returns an integer, and the meaning of return value: Return value is:

Greater than Zero: current Object 是大于被比较的

Less than Zero: current instance is 小于being compared with.

Is Zero: current instance 是与 被比较的一样

你也可以invoke CompareTo() method 作为return type, CompareTo() method is already implemented on integer type, so we can invoke this method and return it’s value.

比如想按class 中的integer,string, char 排序，可以用他们的的CompareTo

public class Customer: IComparable<Customer>

{

public int ID { set; get; }

public int Salary { get; set; }

public string Name { get; set;

}

public int CompareTo(Customer other) { //Other 是别的class

if (this.Salary > other.Salary)

return 1;

else if (this.Salary > other.Salary)

return -1;

else return 0;

}

}

也可以

public class Customer: IComparable<Customer>

{

public int ID { set; get; }

public int Salary { get; set; }

public string Name { get; set;

}

public int CompareTo(Customer other) { //Other 是别的class

return this.Salary.CompareTo(other.Salary);

}

}

还可以自己定义一个IComparer的interface

public class SortByName : IComparer<Customer> {

public int Compare(Customer x, Customer y) {

return x.Name.CompareTo(y);

}

}

在main 中

SortByName sortByName = new SortByName();

l.Sort(sortByName);

Lesson 87 MultiThreading in C#

What is a Process:

Process is what the