Наследование (Inheriting):

```
public class A1
{
    public string F1 { get; set; }
    public A1()
    {
        F1 = "Some default string";
    }

    public A1(string f1)
    {
        this.F1 = f1;
    }
}

class A2 : A1
{
    public string F2 { get; set; }

    public A2(string f2, string f1) : base(f1)
    {
        F2 = f2;
    }

    public A2() /*: base()*/
    {
        F2 = "Another string";
    }
}
```

NullReferenceException

```
class A
{
    public string str1; /*row1*/
    /*row1 Equvivalent to  public string str1 = (string)null;*/
    /*row1 Equvivalent to  public string str1; public A(){ str1 = (string)null; }*/

    {
        Console.WriteLine(new A().instStr1.Length); // System.NullReferenceException
    }
}
```

Классы File и FileInfo

`File` – статический класс. `FileInfo`, создаём объект и применяем объектные методы

Точно также для папок `Directory` и `DirectoryInfo.`

```
File.Copy();
File.Exists();
File.AppendAllText(filePath, "\n56");
File.Create();
Directory.Exists();
Directory.GetFiles();
Directory.GetDirectories();
Directory.GetCurrentDirectory();

public static class File {…}
public static class Directory {…}
public static class Path {…}

Path.Combine();
```

Пример:

```
var fileName = Path.Combine("Y:\\", "newfile.txt");
FileInfo fInfo = new FileInfo(fileName);
var writer = fInfo.AppendText(); --дописать в файл и(!) создать, если не был создан до.
writer.WriteLine("test_line3");
writer.WriteLine("test_line4");
writer.Dispose();
```

Журналирование

```
string eventLog = "Application";
string eventSource = "Logging Demo";
if (!EventLog.SourceExists(eventSource))
    EventLog.CreateEventSource(eventSource, eventLog);

EventLog.WriteEntry(eventSource, "Application started");
```

Сериализация объекта.

```
using System.Runtime.Serialization.Formatters.Binary;

ServiceConfigutation config = new
    ServiceConfigutation();

IFormatter formatter = new BinaryFormatter();
FileStream buffer = File.Create ("filepath");

formatter.Serialize(buffer, config);
buffer.Close();
//////////////////////////
//Десериализация

IFormatter formatter2 = new BinaryFormatter();
FileStream buffer2 = File.OpenRead ("filepath");

ServiceConfigutation config2 =
    formatter2.Deserialize(buffer2) as ServiceConfigutation;
buffer2.Close();
```

Пример: Экземпляр и ссылка на экземпляр.

```
char[] sch1 = new char[] { 'a', 'a', 'a', 'a', 'a'};
char[] sch2 = sch1;
for (int i = 0; i < sch2.Length; i++)
    sch2[i] = 'b';
for (int i = 0; i < sch1.Length; i++)
    Console.WriteLine(sch1[i]);
/*
Output:
b
b
b
b
b
*/

char[] sch1 = new char[] { 'a', 'a', 'a', 'a', 'a'};
char[] sch2 = sch1;
sch2 = new char[] { 'b', 'b', 'b', 'b', 'b' }; /*Теперь ссылка «sch2« указывает на другой(!)*/
for (int i = 0; i < sch1.Length; i++)          /*экземпляр а через ссылки на первый экземпляр
    Console.WriteLine(sch1[i]);                /*мы ничего не меняли */
/*
Output:
a
a
a
a
a
*/
```

Dependency injection

Чем хорошо DI помимо основного свой назначения: он на этапе компиляции отработал и всё. Проект-прога собралась, работает, не тормозит!

Http Request, Response

```
var uri = "http://asd.com";
var request1 = WebRequest.Create(uri) as HttpWebRequest;
var dataReq1 = Encoding.Default.GetBytes(
    "{\"field1\":\"val@1\"}"
    );
request1.Method = "POST";
request1.ContentType = "application/json";
request1.ContentLength = dataReq1.Length;
//request1.Credentials = new NetworkCredential("userName","secretPasswd"); //Для авторизации на
сервере
var dataStream = request1.GetRequestStream();
dataStream.Write(dataReq1, 0, dataReq1.Length); //!
dataStream.Close(); //!


var response = request1.GetResponse() as HttpWebResponse;
var stream = new StreamReader(response.GetResponseStream());
//stream.ReadLine();
stream.Close();
```

http – hypper text transport protocol

Methods

Extension method-ы - method-ы, чтобы воткнуть их в существующий класс, без изменения кода этого класса.(и без создания нового класса-наследника)

```
public static class StatCl
{
    public static bool IsGoodPassword(this string s)
    {
        return s.Length > 7;
    }
}
```

//Опциональные аргументы метода:

public void ExampleMethod(int required, string optionalstr = "default string", int optionalint = 10){ }

//Интересный пример:

```
public static void DatesToPeriodConverter(DateTime start, DateTime end = DateTime.MinValue, out
string date, out string time) { } // - incorrect

// - correct
  public static void DatesToPeriodConverter(DateTime start, DateTime? end = null,
     out string date, out string time)
{
    var effectiveEnd = end ?? DateTime.MinValue; // и т.д.
}
```

Метод можно сделать abstract - просто заглушка без реализации(Тогда и весь класс придётся сделать abstract)

```
abstract class Foo1
{
    public abstract void AbstrMet();
}
```

abstract метод считается автоматом virtual

```
abstract class Foo2: Foo1
{
    public abstract override void AbstrMet();
}
```

///out , ref аргументы.

//ref должны инициализировать до вызова

```
static void test(ref int a)
{
    a++;
    Console.WriteLine("test a = {0}", a);
}
```

//out аргумент можно объявить, без инициализации

```
static void test2(out int a)
{
    a = 10;
    a++;
    Console.WriteLine("test a = {0}", a);
}
```

Задачи типа Task, Асинхронные методы.

Thread- ы,
TPL с 4.0 Framework – Task, задача. параллельная основному потоку.
с .Net Framework 4.5 C#: async/await

```
Task task1 = new Task(new Action(MyMethod));
Task task2 = new Task(delegate { MyMethod(); });
Task task3 = new Task(() =>  MyMethod());
task1.Start(); //Задача стартовала!! Параллельно!!
/////
task1.Wait(); //Хотим зачем-то дождаться
Task.WaitAll(task1, task2, task3);//Хотим дождаться всех

Task<string> task4 = Task.Run<string>(()=>DateTime.Now.ToString());
var result = task4.Result;

CancellationTokenSource cts = new CancellationTokenSource();
CancellationToken ct = cts.Token;

Task.Run(()=> DoWork(ct));

static void DoWork(CancellationToken token)
{
    //...
    if (token.IsCancellationRequested)
    {
        //Closing...
        throw new OperationCanceledException();
        //return;
    }
}
```

```
double[] dArr = new double[50000];
Parallel.For(0, 50000, l =>
{
    dArr[l] = Math.Sqrt(l);
});

var coffeeList = new List<Coffee>();
foreach (Coffee coffee in coffeeList)
{
    CkeckAvailability(coffee);
}
//Эквива
Parallel.ForEach(coffeeList, l => CkeckAvailability(l));


var strongCoffees =
    from coffee in coffeeList.AsParallel()
    where coffee.Strength > 3
    select coffee;


{
Task<string> firstTask = new Task<string>(()=> { return "Hello!"; });

Task<string> secondTask = firstTask.ContinueWith((fres) =>
{
    return String.Format("{0}, World!", fres.Result);
});

firstTask.Start();
```

```
Console.WriteLine(secondTask.Result);
}
```
ОСнова: выигрыш при параллельности может быть ддостигнут ТОЛЬКО на многоядерном процессоорах.
List Exception-в, перебрать и добраться до изначльного Exceptiona.

```
try
{
    task1.Wait();
}
catch (AggregateException ex)
{
    foreach (var inner in ex.InnerExceptions)
    {
        //Handle each exception in turn
    }
}
```
_____

```
private async void btnLongOperation_click(object sender)
{

    Task<string> task1 = Task.Run<string>(()=> {
        //...
        return String.Format(""); });
    lblResult.Content =  await task1;//Это неееe
    //lblResult.Content = task1.Result;
    //Это вернуться в эту строчку и записать Result в lblResult.Content
}

private async Task<string> GetData()
{
    var result = await Task.Run<string>(
        () => {
            //.... Thread.Sleep(10000);
            return "Operation Complete.";
        }
        );
    return result;
}

private async void btnGetData_click(object sender)
{
    lblResult.Content = await GetData();
}
```
«В отдельном потоке»
Обработчик в основном потоке.
Основной поток не блокируется.

```
Customer customer = new Customer();
var widgetsTask = _widgetService.GetAllWidgets();
var foosTask = _fooService.GetAllWidgets();
customer.Widgets = await widgetsTask;
customer.Foos = await foosTask;

return customer;
```

LINQ

```
string[] mstr = { "ab", "cd", "cd" };
List<string> lstr = new List<string>();
lstr.Add("ab");
lstr.Add("cd");
lstr.Add("de");
lstr.Add("cd");
lstr.Add("cd");

var query =
    from el in lstr
    where el != "ab"
    select el;

var query2 =
    from el in lstr
    join em in mstr on el equals em
    select el;

var query3 =
    from el in lstr
    where el != "ab"
    group el by el into g
    select /*new { a =*/ g.Key /*}*/;

foreach (var e in query3)
{
    Console.WriteLine(e);
}
//////////////////////
var query1 = from char c in charMass
             group c by c into g
             select new { a = g.Key, b = g.Count() }; // Создаём анонимный объект. Нать хоть как-то
обозвать, просто не возьмёт
//////////////////////

var custSupJoin =
    from sup in suppliers
    join cust in customers on sup.Country equals cust.Country
    select new { Country = sup.Country, SupplierName = sup.SupplierName, CustomerName =
cust.CompanyName };

var categories =
    from prod in products
    group prod by prod.Category into prodGroup
    select new { Category = prodGroup.Key, TotalUnitsInStock = prodGroup.Sum(p => p.UnitsInStock) };
```

Constraints

```
//public static interface ISt // static нельзя в интерфейсах.
//{
//}

//public interface ISt
//{
//    static void MStat(); //static нельзя в интерфейсах.
//}

//static class ClassSt2 /*: ClassSt2 */ // Наследование только от System.Object
//{
//}
```

```
public class CustomList<T> where T : ICollection { }
```
// Теперь можно типизировать эту коллекцию только классами (ИЛИ Структурами) реализующими интерфейс `ICollection`.
public class CustomList<T> where T : IBeverage, IComparable<T>, new(), struct (только структурами), class (только классами)

ГЛАВНЫЙ ПРИНЦИП ПРОГРАММИРОВАНИЯ:
КАКОЕ-ТО РЕШЕНИЕ, НО РЕШЕНИЕ -
НЕСРАВНИМО ЛУЧШЕ СУПЕРКРАСИВОГО
МЕЧТАНИЯ НЕ В КОДЕ