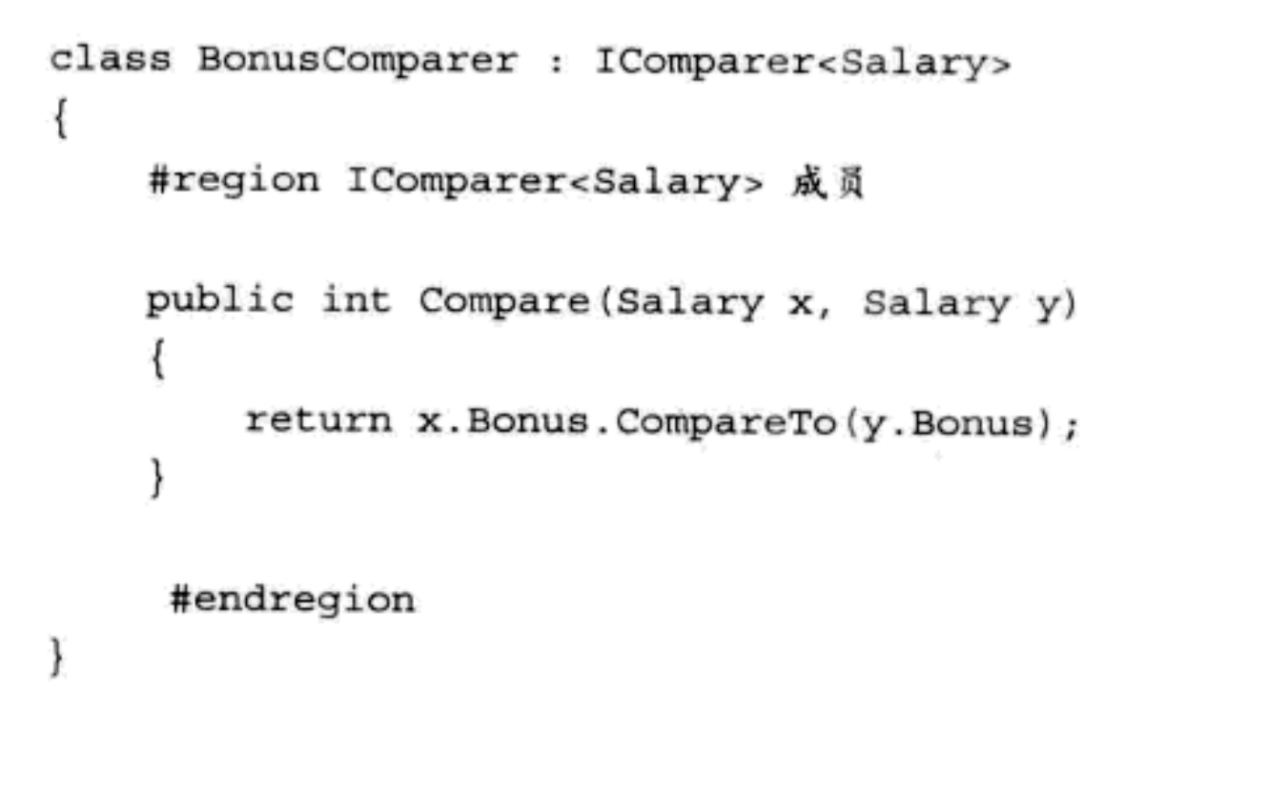
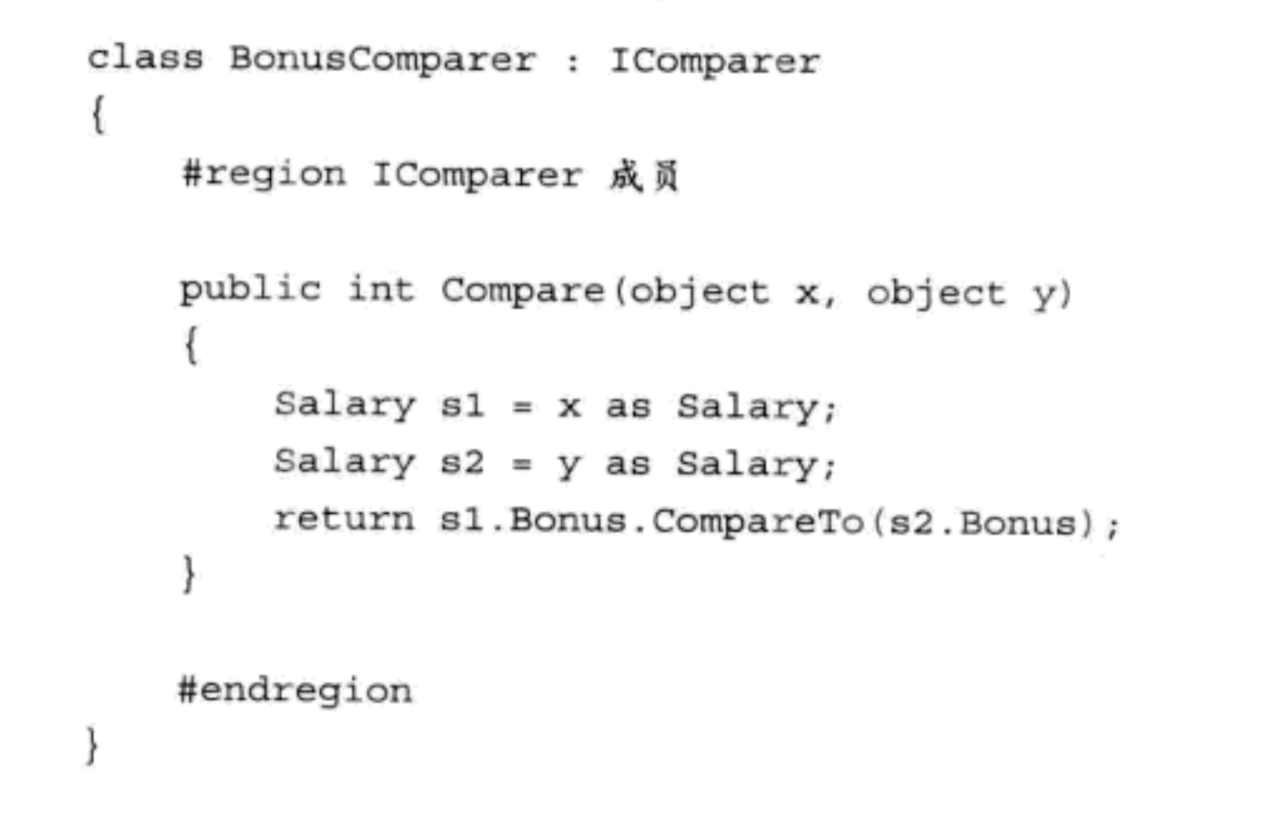
# C#高质量代码

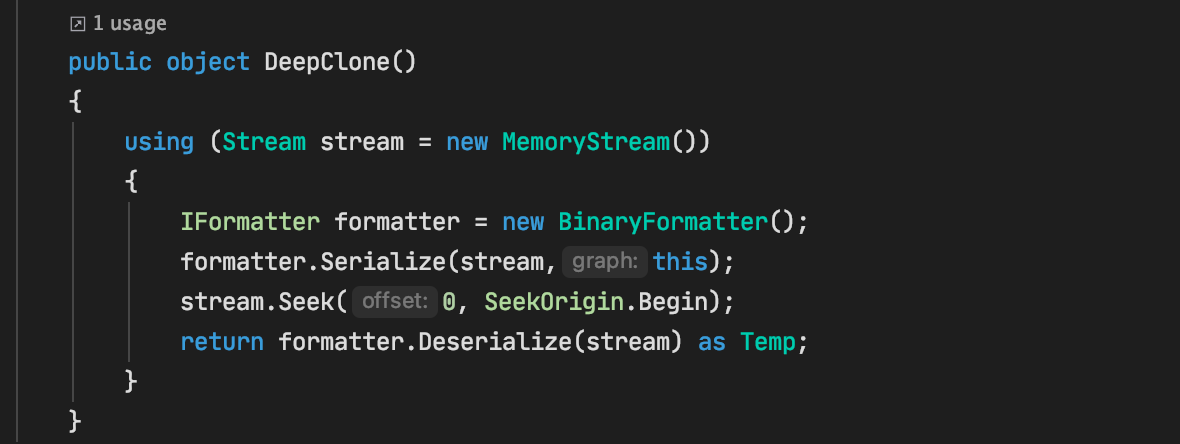
### 第一章：基本语言要素

1. 正确操作字符串：
   1. 与数字相加，使用 num.tostring() 可减少一次装箱
   2. 字符串使用+ 和 = 时会都会在内存中产生一个新的字符串对象，建议一次赋值或者拼接次数过多使用StringBuilder
2. 使用默认转型类型
   1. 使用强制转换和隐式转换
   2. 使用内置parse，trypares 或如tostring，todouble...
3. 区别对待强制转换与as和is
   1. 一个类实现了强转另一个类的方法则必须使用强转型，使用as会报错。基类转子类应该使用as，子类与子类之间转换，则应该提供转换操作符，以便强转
   2. as比强转型效率更高。强转会抛出异常导致性能降低。
   3. is用于判断。
4. TryParse比Parse更好
   1. TryParse返回true和false Parse则会进行异常捕获对性能进行耗损
5. 使用T？来确保值类型也可以为null
   1. T？为Nullable<T>的缩写
   2. Int? I = 123; int j = i??0; 可视为 i!=null ? i:0;
6. 区别对待const和readonly
   1. const为编译期常量，readonly为运行时常量.const效率更高，readonly类型更为灵活
   2. const只能修饰值类型和string，readonly没有限制
   3. const为编译期常量，所以它天然为static类型，不能再用static进行修饰
   4. readonly修饰值类型时，值本身无法被改变。修饰引用类型时，引用的指针无法改变，即引用的对象的值还是可以被改变的。
   5. readonly在构造方法中可以多次赋值。
7. 将0值作为枚举类型的默认值
   1. 未赋值的枚举 默认值为0 （可通过static T 进行查看）
   2. 将一个枚举中没有的int类型强转为枚举类型不会报错
8. 避免给枚举类型的元素提供显式值
   1. 枚举类型会将没赋值的默认赋值为 前一个枚举+1,显式赋值可能会导致不同的枚举指向同一个值 (eg: one =1 four = 4, three, five =5 此时three和five都为5)
   2. 本建议也有例外，如需要进行亦或操作时，可以进行显式赋值（1,2,4,8,16...）
9. 习惯重载运算符
   1. 在某些地方，如数学相关计算，使用运算符比方法更便于理解和符合习惯。
10. 使用比较器使用泛型而不是非泛型
    1. 可以看出非泛型的比较器会进行类型转换，这是会对性能进行影响的
11. 区别对待 == 和 Equals
    1. == 和 Equals都倾向一个原则：对于值类型，如果值相等，则应该返回True。对于引用类型，如果指向同一个对象返回True
    2. Equals可以进行重载来表示引用类型的值相等
12. 重写Equals方法时，将其当成值使用时也要重写GetHashCode
    1. 在使用Dictionary类时，其使用hashCode进行索引，没重写GetHashCode则会调用object的默认GetHashCode。object每new一个对象则会对其生成一个固定的整数值，默认方法会对其进行求hash.(整数求hash为其本身)
13. 正确深拷贝和浅拷贝
    1. 浅拷贝：值类型的值拷贝，引用类型的引用拷贝。对引用类型修改会影响到源对象

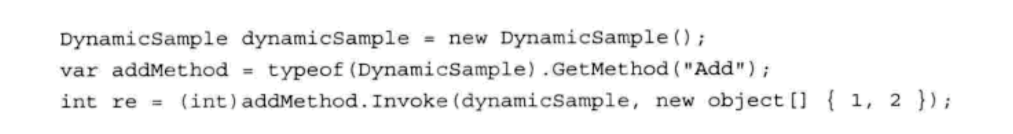
深拷贝：值类型的值拷贝，引用类型将引用类型的字段都重新赋值到新的对象里面，对该对象修改不会影响到源对象

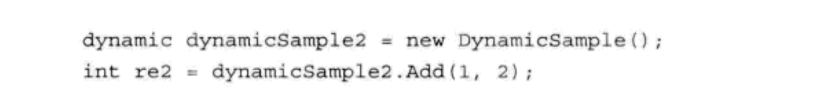
b) 浅拷贝可以用Iclone中的MemberwiseClone方法

c) 深拷贝可以使用序列化，防止手动拷贝时漏掉属性。代码如下：



1. 使用dynamic来简化反射实现
   1. var和dynamic不同，var是编译期语法糖，编译期会自动匹配类型。dynamic在被编译后仍然为object类，只是编辑器会对其进行特殊处理
   2. dynamic与反射区别，减少拆箱操作，简化代码。

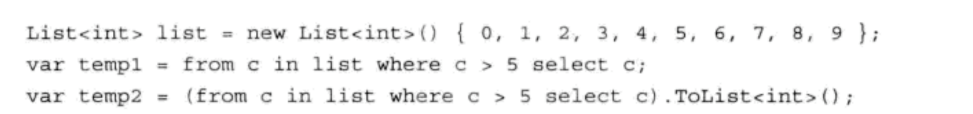
正常反射代码：

dynamic代码：

### ：集合和LINQ

LINQ（Language integrated Query，语言集成查询） 提供了类似SQL的语法

1. 元素数量可变的情况下不应该使用数组，使用List
   1. C#中只有Array和List List可动态添加和减少。array则固定大小，给其扩大size时需要重新分配内存以及拷贝数据
   2. C++中有Vector (C# 中List原理一样) 该数据结构在扩大内存时，会创建一个新的数组 大小为之前数组的2倍，然后将原始数组的数据进行拷贝并改变原始指针的指向。
2. 大多数情况下使用foreach
   1. 集合都实现了IEnumerable接口，foreach遍历时读取集合的Enumerator元素进行遍历，不同的集合可以使用统一的foreach进行遍历。
   2. foreach会自动进行异常捕获,对象有Dispose（资源释放）方法时。会自动在最后进行调用Dispose方法
   3. 代码简洁
3. foreach不能代替for
   1. foreach只能遍历不能对遍历的元素和集合进行修改，不能增删改。
   2. foreach和for的底层都是对一个泛型数组进行访问。但foreach循环时会对集合的版本号进行判断，任何增删改都会使集合版本+1，导致判断出错。
4. 使用泛型集合代替非泛型集合
   1. 泛型集合与非泛型相比少了装箱和拆箱操作，效率更高
   2. 泛型比非泛型更安全
5. 使用正确的集合
   1. 直接存储类型：Array(数组，List) 特点：寻找修改和添加速度快，插入和删除速度慢
   2. 当集合的数目固定且不涉及到转型，使用数组效率更高，否则用List<T>
   3. 根据数据特点选择数据结构，如列表，stack，queue
6. 迭代器应该是只读的
   1. 如果迭代器是可写的会导致其他访问迭代器的访问到错误的迭代器
7. 理解延迟求值和主动求值区别
   1. 延迟求值：当查询时再访问数组，并在每次访问时重新查询
   2. 主动求值：立即查询，并在之后不会随着主体变化而变化
      1. Temp1为延迟求值 temp2为主动求值



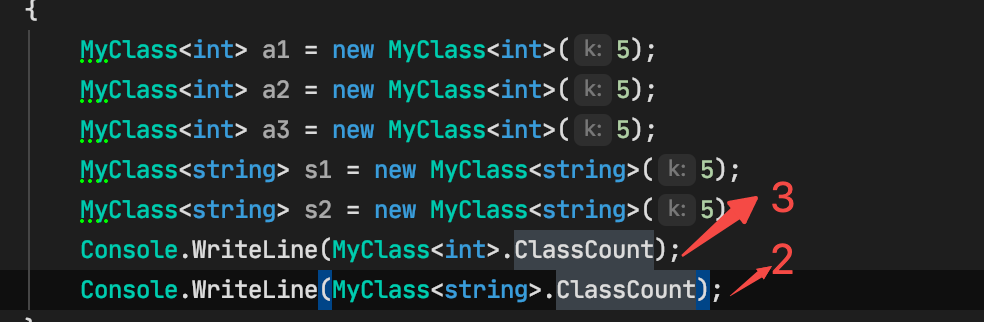
### 第三章：泛型、委托、和事件

1.创建类时优先考虑泛型。

A.与正常类相比，加强了代码重用性

B.与object类相比，类型安全+减少拆箱和装箱消耗

1. 避免在泛型中使用静态变量
   1. 泛型类根据不同的数据类型生成不同的代码，也就是说只有相同的数据结构才能共享静态变量

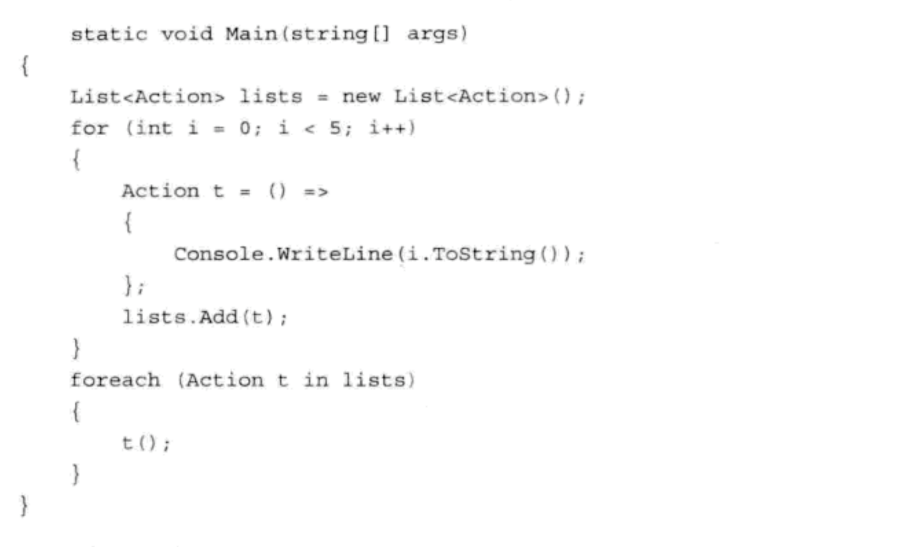


1. 为泛型限定约束 （Where T: XXX）
   1. 使用约束后可以使用约束类型的属性和方法，将其当做一个类来使用而不是一个object
2. 使用default为泛型变量指定初始值
   1. 泛型为类时，初值为null。值类型，初值为0。当不确定泛型类型时手动赋值会报错编译不通过。使用default会自动匹配类型并返回正确的值
3. 使用FCL中的委托声明
   1. FCL有三类委托声明：Action，Func，Predicate
   2. Action表示接受0个或多个参数，没有返回值。
   3. Func表示接受0个或多个参数，带返回值，最后一个为返回值类型。

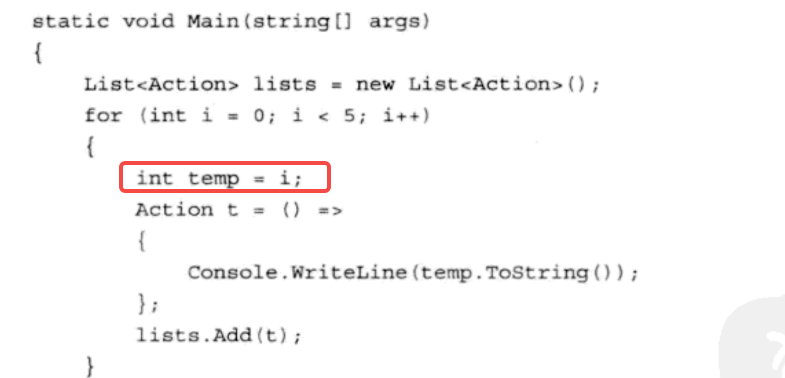
Func<Result>, Func<T,Result>,Func<T,T2,Result>

d) Predicate:public delegate bool Predicate<int T>(T obj) 用来判断一个对象是否符合标准。 经常用于Fund方法。 Array.Find()

1. 使用Lambda表达式代替简易方法（ func = (a,b) =>{xxx} ）
   1. 不用在外部声明方法。
   2. 简化代码
2. 小心闭包中的陷阱
   1. 匿名函数引用了某个局部变量，编译期会将其提升到闭包对象中,即将for循环中的变量i修改成了闭包对象的公共变量i eg:下图输出对象都为5



b)解决方法：将闭包对象的产生放在for循环内部



1. 使用泛型参数兼容泛型接口的不可变性
   1. Isalary<Child> 不能当成Isalary<Parent>传入方法内。

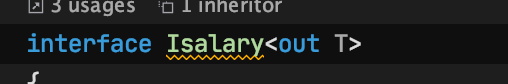


方法可以使用泛型来接受参数

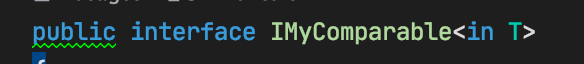


b) 也让接口中的泛型支持协变（协变：让子类可以当成父类传入）

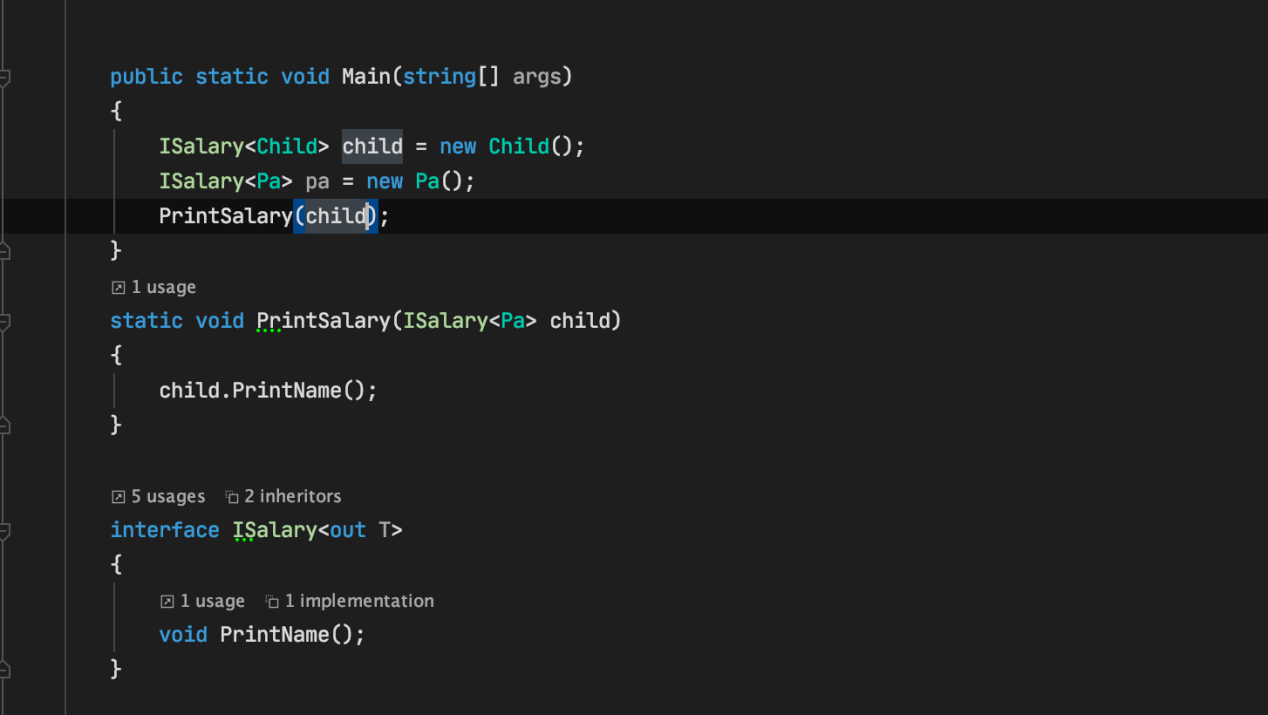
在接口中泛型前加out关键字



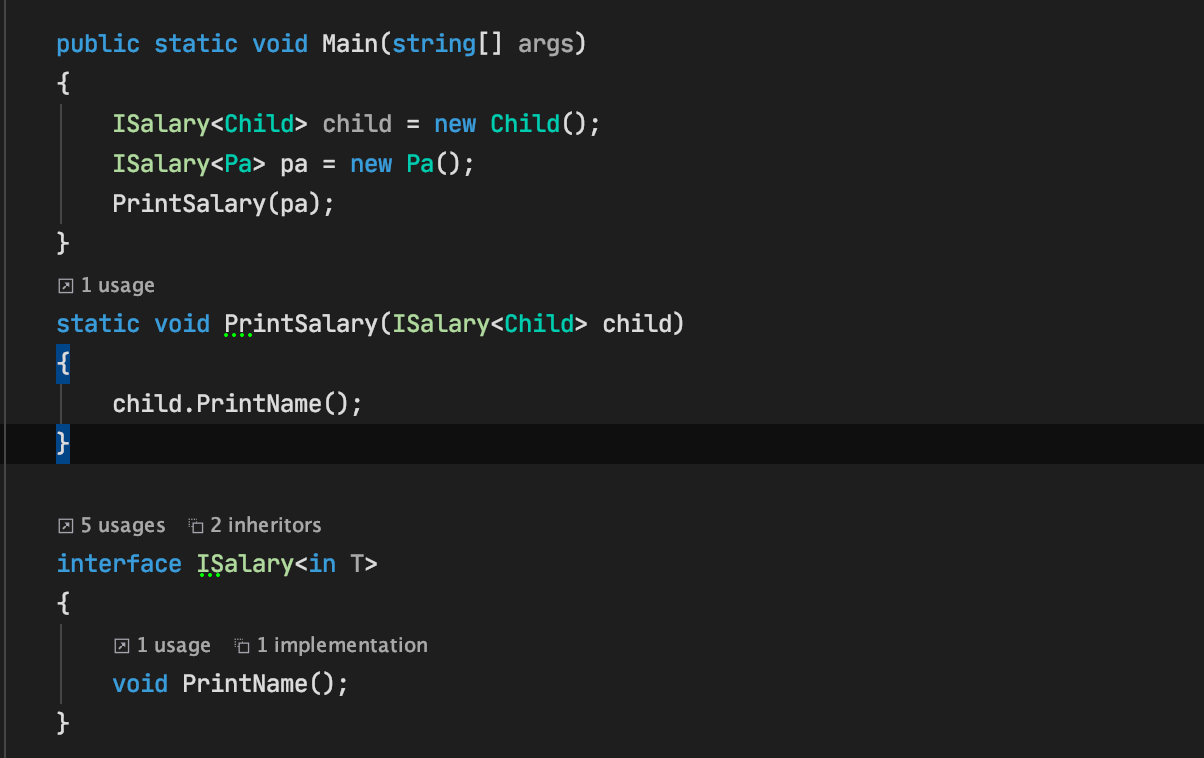
c) 在接口中加入in关键字可以让泛型支持逆变（逆变：让父类可以当成子类传入）



协变：



逆变：

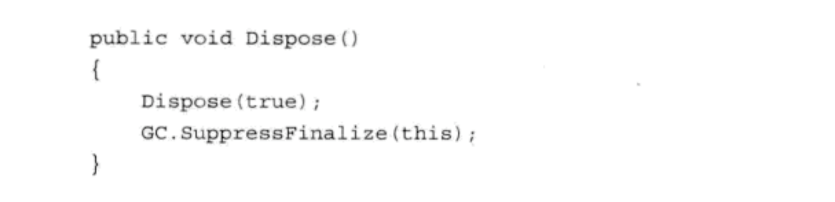


### ：资源管理和序列化

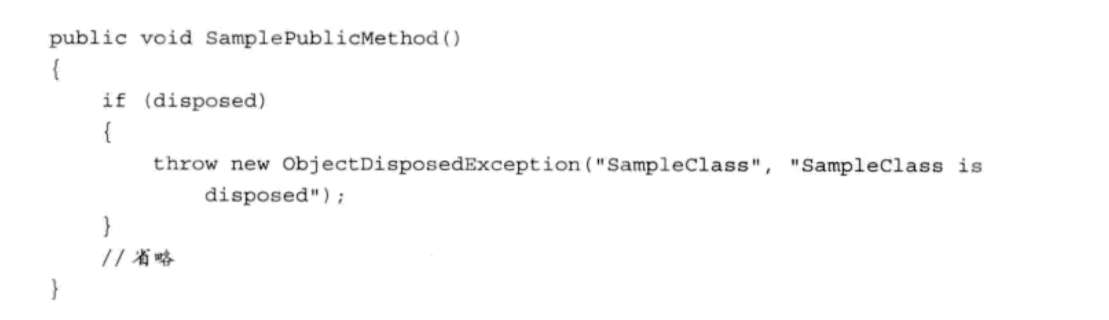
1.显式释放资源需要继承接口IDispose

* 1. 非托管资源需要手动释放
  2. 托管资源 CLR会在GC时进行资源释放，程序员不知道何时进行释放。需要手动释放资源时需要继承IDispose
  3. 即需要显式的释放资源就需要继承接口IDispose

1. 及时继承了IDispose方法，也需要在析构方法中提供隐式清理
   1. 即使玩家继承了IDispose方法，GC在清理回收该内存时，只会调用析构函数并不会直接调用Dispose方法，基于此，我们应该在析构中方法中提供隐式清理，以避免调用者忘记调用该方法导致资源泄漏
2. 调用者显式调用Dispose后，那么析构方法也就没必要执行了。
   1. 手动释放了资源也就不需要GC时再隐式的释放资源。 GC.SuppressFinalize可以告知GC不再调用析构方法

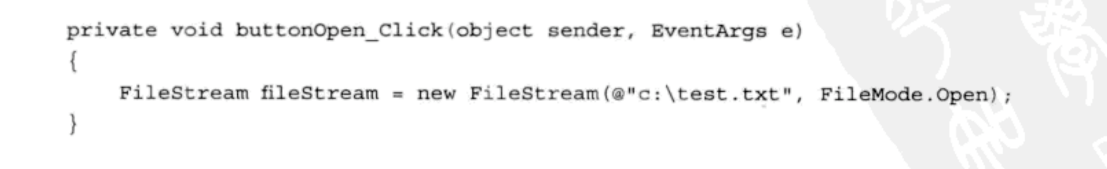


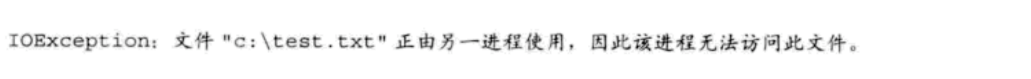
1. Dispose方法应该支持多次调用
   1. 因为调用Dispose之后，该对象并没有被置为null，对象的指针还是存在，只是内部数据已经被销毁了。
   2. 在Dispose之后，调用该对象的公共方法时应该告知调用者该对象已被Dispose



1. 在Dispose模式中应该提供一个受保护的虚方法
   1. 在子类继承该Dispose时，该虚方法用来提醒子类也该清理父类的Dispose。

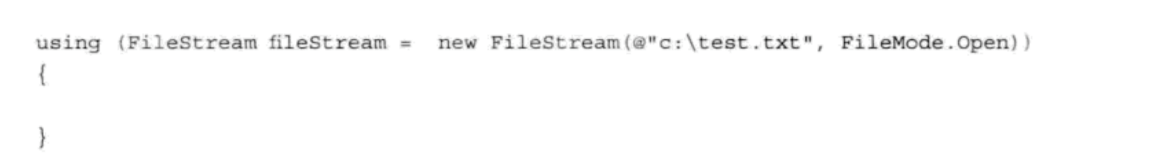


1. 在Dispose方法中，应该区别对待托管资源和非托管资源
   1. 在显示调用中，Dispose方法应该清理掉托管资源和非托管资源
   2. 在隐式调用中，即析构方法中，Dispose应该只清理掉非托管资源，而托管资源则交由垃圾回收器自己处理。
2. 及时释放资源
   1. 



该方法为点击一个按钮，打开test.txt文件。如果点击两次按钮，则会IO报错。原因是此时fileSteam虽然没有再被引用，但GC还没有到回收的时机（内存不足，定时回收），导致该文件一直占用直到GC调用。

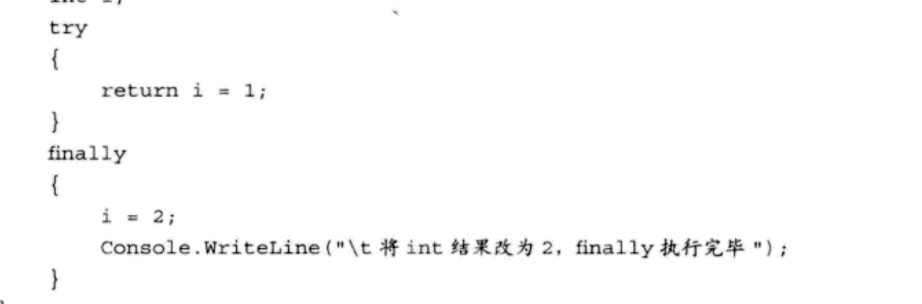
可调用DIspose方法在fileSteam用完后释放掉。也可以使用语法糖using



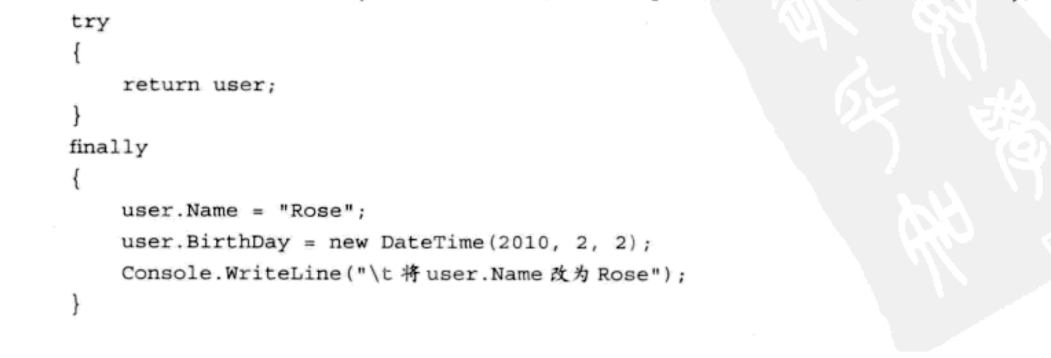
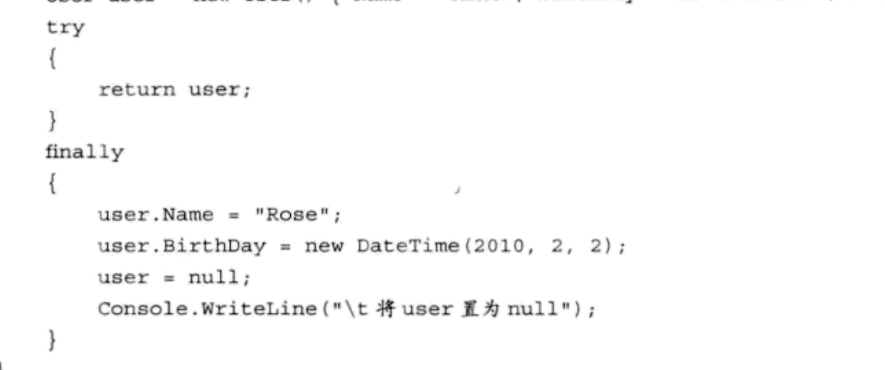
1. 静态变量后续都不会使用时将其设置为null
   1. 静态变量一旦创建就会一直存在，垃圾回收器不会将其视为垃圾，当你后续不再继续使用时将其设置为null回收其资源。

9.不需要序列化的变量 特性设置为【NonSerialized】节省开支

### ：异常与自定义异常

1. 注意finally的作用域
   1. finally中的代码始终会被执行，即便在try里面执行了return.finally中代码还是会被执行
   2. 

返回结果1

* 1. 返回结果 “Rose”
  2. 

返回结果“Rose” user = null 相当于user指向null return user 返回的是user的地址

2.

# 第二部分：架构篇

### ：成员设计

1. 谨慎将数组或集合设置为属性
   1. 数组设为只读属性，外部不能修改数组大小但可以随意修改数组元素的内容
2. 不要在父类构造函数中调用虚方法
   1. 子类继承了父类并实现了虚方法，在初始化时。会先调用父类的构造方法，然后再执行子类的初始化。在调用父类的构造方法时，由于是虚方法会调用到子类的方法，此时子类还未完成初始化可能导致错误
3. 不同组织的应用场景
   1. 接口：对象关系并不紧密，但是功能拥有相同声明。更多的是“Can do”。典型例子：IClone
   2. 抽象类：对象关系密切且功能相近。更多的是“Is a”。典型的例子：Person
   3. 组合：可以组合多个其他类型。更多的是“Has a”。典型的例子：Controller
4. 为静态类添加静态构造函数
   1. 好处：在调用类成员之前被执行。可以初始化静态成员并捕获该过程发生的异常。
5. 单例和静态类区别
   1. 单例作为实例对象，可以被当成参数进行传输和返回，静态类不行
   2. 单例可以进行继承和实现接口。静态类不能进行继承和实现借口
6. 当枚举具有更多的行为时可以考虑将其用类来实现
7. MD5安全性
   1. MD5： R = H（S） S到R是一个多对一的映射，H是一个压缩算法，R的空间远小于S。 S不能通过逆方法进行还原
   2. MD5虽然不能被逆运算，但是可以通过枚举法进行破解
   3. 更安全的做法即是对MD5进行改进，一般的做法是通过多次使用MD5增加值的复杂度
8. 对称算法加密和非对称算法对比
   1. 对称算法：接收方和发送方规定一个秘钥或一对，双方通过该秘钥进行解密和加密
      1. 解密快，系统开销快
   2. 非对称算法：一对秘钥，加密方发送数据时会将公钥一并发送，接收方通过秘钥进行解析。
      1. 解密的密钥不需要发送给对方
      2. 设置复杂，解密开销比较大。适用于数据量小的情况

# 第三部分：编码规范及习惯

### ：命名规范

1. 用名词给类型命名 ：OrderProcessor
2. 用形容词给接口命名：IDisposable
3. 考虑让派生类以基类名为后缀名
4. 泛型参数要以T作为前缀
5. 以复数命名枚举类型，以单数命名枚举元素 Week：Day 这里week作为复数，day作为单数
6. 使用肯定性短语命名Bool类型。Is Can Has作为前缀。 IsActive HasChilde